

# Akribis-Agito Controller

## Keywords Reference

## Table of contents

---

<b>General</b>	<b>18</b>
<b>Using User Units</b>	<b>19</b>
General	19
Calculations details	19
<b>Keywords Reference</b>	<b>22</b>
AAmpFullScale	23
Abort	24
About	25
AbsTrgt	26
Accel	27
AccelFact	28
AccFFW	29
AccShapeDist	30
AccShapeFact	31
AccShapeOn	32
AlnDB	33
AlnFilt	34
AlnGain	35
AlnMode	36
AlnOffset	37
AlnPort	38
AllStat	39
AmpType	41
AntiCoggAmp	42
AntiCoggOn	43
AntiCoggPhase	44
AntiCoggValue	45
AOutMode	46
AOutOffset	47

AOutPort	48
AOutShifts	49
AutoExec	50
AutoGAccTh	51
AutoGBW	52
AutoGCopy	53
AutoGDownSm	54
AutoGJm	55
AutoGJratUs	56
AutoGKt	57
AutoGMask	58
AutoGMaxLen	59
AutoGMaxRat	60
AutoGMinLen	61
AutoGMinRat	62
AutoGMode	63
AutoGNumSet	64
AutoGOn	65
AutoGPosFilt	66
AutoGQualTh	67
AutoGSavPer	68
AutoGStatus	69
AutoGVelTh	70
AuxEncDir	71
AuxEncFilt	72
AuxEncSubType	74
AuxEncType	75
AuxIndexPos	76
AuxIndexStat	77
AuxLockCntr	78
AuxLockEn	79
AuxLockSrc	80
AuxLockVal	81

AuxPos	82
AuxUsrUnits	83
AuxVel	84
Begin	85
BeginDInOn	86
BeginOnToPos	88
BrakeLockTime	89
BrakeMode	90
BrakeRelTime	91
BrakeUsed	92
CalcFilters	93
CalcIden	94
CANAddr	95
CANBaud	96
CANDelay	97
ChainAddress	98
CIAutoConnect	99
CIConnect	100
CIDeviceType	101
CIDisconnect	102
CIGlobalStat	103
CIIdentity	104
CIOfflineData	105
CIOfflineDef	106
CiOfflineSend	107
CIStatus	108
CISyncDef	109
ClearErr	110
ComtAng	111
CNCAAbsTrgt	112
CNCAAccel	113
CNCAClear	114
CNCADecel	115

CNCADoStep	116
CNCAdPosRef	117
CNCAEmrgDec	118
CNCAEncRatio	119
CNCAEndSpeed	120
CNCAFIFO	121
CNCAJerk	122
CNCAPause	123
CNCAPercents	124
CNCAPosRef	125
CNCAPushParam	126
CNCAPushSeg	127
CNCAPushType	128
CNCARemove	129
CNCASpeed	130
CNCAStatus	131
CNCAStepMode	132
Compare	133
ComtAng	134
ComtMode	135
ComtStatus	138
ConFlt	139
ContCL	141
ControlMode	142
CounterDown	143
CounterUp	144
CurrAlnTh	145
CurrCmdCntr	146
CurrCmdIndex	147
CurrCmdSrc	148
CurrCmdTime	149
CurrCmdVal	150
CurrDir	151

CurrGain	152
CurrKi 153	
CurrLimFwd	154
CurrLimMode	155
CurrLimRev	157
CurrPosErrTh	158
CurrRef	159
DCDC	160
DebugData	161
Decel	162
DInFilt	163
DInLog	164
DInMode	165
DInPort	172
DoNothing	173
DOutLog	174
DOutMode	175
DOutPort	178
DOutPortCBit	180
DOutPortSBit	181
DOutPortTBit	182
DOutSelect	183
DOutType	185
DownloadFW	186
DownloadUPBin	187
DPosRef	188
DualLoopFact	189
DualLoopOn	190
DualStuckTime	191
DualStuckVel	192
DynBrakeOn	193
DynBrakeSpeed	194
ECAMCycCount	195

ECAMCycles	196
ECAMEnd	198
ECAMEndCyc	199
ECAMGap	200
ECAMMaster	201
ECAMMasterIni	202
ECAMStart	203
ECAMStartCyc	204
ECAMTableNum	205
EmrgDec	206
EmulFilter	207
EmulIndexType	208
EmulRat	209
EncDir	210
EncFilt	211
EncRes	213
EncSubType	214
EncType	215
EPPDenFract	216
EPPDenInteg	217
EPPFiltLength	218
EPPModelRange	219
EPPNumFactor	220
EPPNumFract	221
EPPNumInteg	222
EPPRequest	223
EPPState	224
ErrLog	225
EthernetIP	229
EthernetMAC	230
EthernetPort	231
EventBegPos	232
EventCntr	233

EventEndPos	234
EventGap	235
EventNextPos	236
EventOn	237
EventPulseWid	238
EventSelect	239
EventTable	240
EventTableBeg	241
EventTableEnd	242
EventTableSel	243
EventType	244
FastIdDownSam	245
FastIdInit	246
FIFOClear	247
FIFOCycleTime	248
FIFOPushCycle	249
FIFOPushLinP	250
FIFOPushLinV	251
FIFOPushParA	252
FIFOPushParP	253
FIFORemove	254
FIFOStatus	255
FIFOType	256
FIFOValue	263
Force	264
ForceAlnTh	265
ForceCmdCntr	266
ForceCmdHTime	267
ForceCmdIndex	268
ForceCmdSlope	269
ForceCmdSrc	270
ForceCmdVal	271
ForceErr	272



ForceFFW	273
ForceGain	274
ForceKd	275
ForceKi	276
ForcePosErrTh	277
ForceRef	278
ForceRefFilt	279
ForceRefFOn	280
ForceVelFFW	281
FrictionComp	282
FwdPLim	283
GantryAccFFW	284
GantryFdbk	285
GantryOffset	286
GantryOn	287
GantryPosGain	288
GantryVelGain	289
GantryVelKi	290
GantryYawRef	291
GenData	292
GoToCurrMode	293
GoToForceMode	294
GoToPosMode	295
HallsAngle	296
HallsValue	297
HomeStat	298
HomingDef	299
HomingOn	300
HomingStat	311
Ia	312
IaErr	313
IaRef	314
Ib	315

IbErr	316
IbRef	317
Id	318
IdenResults	319
Identity	320
IdErr	321
IdRef	322
IndexPos	323
IndexStat	324
IndirectArray	325
IndirectDo	326
IndirectIndex	327
IndirectValue	328
InjectCurrAmp	329
InjectCurrDC	330
InjectedValue	331
InjectForceA	332
InjectFreq	333
InjectPoint	334
InjectPosAmp	335
InjectTimeOn	336
InjectType	337
InjectVelAmp	339
InTargetStat	340
InTargetTime	341
InTargetTol	342
InTargetVelTh	343
Iq	344
IqErr	345
IqRef	346
Jerk	347
Jump	349
LAmpFullScale	350

LAmpVBus	351
LimitsStat	352
Lm	353
Load	354
LockCntr	355
LockEn	356
LockSrc	357
LockVal	358
LockValTable	359
MapEncoder	360
MapLength	361
MapPosGap	362
MapStartIndex	363
MapStartPos	364
MapTable	365
MapType	366
MasterFact	370
MasterFilt	371
MasterPos	372
Math	373
MaxAcc	375
MaxForceErr	376
MaxMotorCurr	377
MaxPhaseCurr	378
MaxPosErr	379
MaxPWM	380
MaxPwrTemp	381
MaxVBus	382
MaxVBusAbs	383
MaxVBusTime	384
MaxVel	385
MaxVelErr	386
MinVBus	387

ModRev	388
MotionMode	390
MotionReason	392
MotionSamples	393
MotionStat	394
MotorCurr	395
MotorOn	396
MotorType	397
OfflineALog	399
OfflineBLog	400
OneOverTOn	401
OpenLoopCurr	403
OpenLoopOn	404
OpenLoopVolt	405
OperationMode	406
PDFact	407
PDFactDen	408
PDFiltFact	409
PDPos	410
PDUsrUnits	411
PDVel	412
PeakCL	413
PeakTime	416
PolePrs	417
PopParam	418
Pos	419
PosErr	420
PosFiltDef	421
PosFiltOn	422
PosGain	425
PosPosFlag	426
PosPosTh	427
PosRef	428

PowerSupply	429
ProgBreaks	430
ProgCallDepth	431
ProgCallStack	432
ProgClrCall	433
ProgClrExp	434
ProgError	435
ProgEventEn	436
ProgEventGEN	437
ProgEventMask	438
ProgEventOn	439
ProgEventPar	440
ProgEventStat	441
ProgEventType	442
ProgEventVal	443
ProgExpDepth	444
ProgExpStack	445
ProgFunc	446
ProgFuncCall	447
ProgHalt	448
ProgHaltAll	449
ProgHaltThis	450
ProgInfo	451
ProgLine	452
ProgPointer	453
ProgPriority	454
ProgReset	455
ProgResetAll	456
ProgRun	457
ProgSingle	458
ProgStat	459
ProgStatAll	460
ProgTask	461

ProtectMask	462
PStatInterval	463
PStatOn	464
PStatParams	465
PStatPort	466
PTPKeepMoving	467
PushConstant	468
PushParam	469
PwrTemp	470
RecData	471
RecGap	475
RecLength	476
RecParam	477
RecStart	478
RecStat	479
RecStop	480
RecTrigForce	481
RecTrigMask	482
RecTrigPos	483
RecTrigSrc	484
RecTrigTyp	485
RecTrigVal	486
RecUpload	487
RefOffsetSamp	488
RefOffsetStep	489
RegenOff	490
RegenOn	491
RegenUsed	492
RelTrgt	493
Reset	494
Return	495
RevPLim	496
RLType	497

Rm	498
RNDDebug	499
RptWait	500
RSBaud	501
Save	502
ScheduleGains	503
ScheduleMode	504
SchedulePos	507
ScheduleSet	508
ScheduleTemp	509
ScheduleTime	510
ScheduleVel	511
SetPDPos	512
SetPosition	513
ShapingDamp	514
ShapingFreq	515
ShapingOn	516
SinCosAIInPort	518
SinCosSetup	519
Speed	520
SpeedChgDir	521
SpeedChgNew	522
SpeedChgOn	523
SpeedChgPos	524
SpringCurrFFW	525
SpringOn	526
SpringPHigh	527
SpringPLow	528
SpringPosFFW	529
StatReg	530
StepBits	534
StepInMotCurr	535
StepInPosCurr	536

Stop	537
StopCNCA	538
StopECAM	539
StopFIFO	540
StopOnHome	541
StopOnIndex	542
StopRep	543
StopVec	544
StuckCurr	545
StuckTime	546
StuckVel	547
Targets	548
Time	549
TorqCompFix	550
TorqCompMode	551
UserMode	552
UserParam	553
UserPWM	554
UserPWMDiv	556
UsrUnits	557
Va	558
Vb	559
VBus	560
Vc	561
Vd	562
VecAbsTrgt	563
VecAccel	564
VecArcCenter	565
VecArcDir	566
VecDecel	567
VecdPosRef	568
VecEmrgDec	569
VecEncRatio	570



VecJerk	571
VecMemberAxes	572
VecMotionStat	573
VecPause	574
VecPosRef	575
VecSpeed	576
VecType	577
Vel	578
VelErr	580
VelFFW	581
VelFiltDef	582
VelFiltOn	583
VelGain	586
VelKi	587
VelRef	588
VLogic	589
Vq	590
WaitStatus	591
WaitTime	593

## General

---

This document provides detailed reference for the communication with AG300 Servo Controller.

It covers the following subjects:

- General communication definitions including:
  - RS-232 (and USB) detailed communication protocol and syntax.
  - CAN Bus detailed communication protocol and syntax.
- Usage of User Units in AG300 Servo Controller.
- Detailed description of the functionality and attributes of each communication keyword.

Additional User's Manuals that are supplied with the AG300 Servo Controller:

- **Hardware User's Manual.**  
Covers: installation, connections, safety and hardware interfaces of the AG300 Servo Controller.
- **Software User's Manual.**  
Covers: Operational modes, motion modes and applicative aspects of using the AG300 Servo Controller.
- **PC Software User's Manual.**  
Covers: description of the PC Software provided with the AG300 Servo Controller.

## Using User Units

---

### General

---

The Servo Controller enables the usage of User Units under the following conditions:

1. The actual encoder (main encoder) has resolution higher than the user units.
2. The ratio between the user units and the encoder counts is an integer value.

For example: If a given encoder has a resolution of 1 [count] = 1 [micron] and the requested user units are 0.05 [mm], then the ratio is  $0.05/0.001 = 50$  (greater than 1 and integer value ... OK).

A ratio smaller than one, or non-integer ratio are not supported by the servo controller.

User Units are used only over the communication. Internally, the servo controller always uses and maintains its readings and control algorithms using the feedback resolution.

Separate ratios may be defined for each of the feedback inputs: Main, Auxiliary and Pulse/Direction, to support different encoder resolutions and different user units for each feedback channel.

The user units are automatically used for all parameters that refer to a position value. This includes parameters like: Position, Velocity, Acceleration, etc.

Special algorithms (see below) are implemented when these parameters are assigned over the communication (in user units), to avoid any possibility for losing position zero readings and/or drifting.

The user can set the User Units factor, which is a software parameter, to 1 to work with the native resolution of the feedback ([counts]).

### Calculations details

---

The position is measured using an encoder, so naturally the controller uses encoder counts in the implementation of the control loops, trajectory calculations and any usage of the feedback position.

The user can select a coarser resolution for the interface by setting a ratio between the physical encoder counts and "user units". For example, the user may be interested in position information every 0.05mm. If the distance of 0.05mm is equivalent, for example, to 10 encoder counts the ratio is set to 10.

## Controller operation scenario using user units:

There are two different types of position counting in the controller:

InternalPosition = the position in encoder counts.

UserPosition = the position reported to the user, rounded to an integer number of "user units"

On power up the controller registers the unknown position of the motor as "InternalPosition = 0".

From now on, any encoder reading between -4 and 5 (for the above example of ratio = 10) will be reported to the user as "UserPosition = 0".

If the motor is turned on without any motion of the motor, it is obvious that any position command from the user will result in an internal command to move to an encoder position that is a multiple of 10. For example, an absolute move to UserUnits = 200 will result in a move to InternalPosition = 2000. This yields a "ruler" or "grid" with marks at InternalPosition = -100, -90 ... -10, 0, 10, 20...

This ruler is our analogy to the encoder requested by the user.

We now can remove the demand to have motor on immediately. Suppose the motor was manually moved to InternalPosition = 3. If the user queries the position he will receive the reply "UserPosition = 0". If the user now turns the motor on, the internal position loop will keep the motor in place using the high-resolution encoder. The motor will be held in "InternalPosition = 3", though the user is not aware of it.

An absolute move to UserUnits = 200 will result in a move to InternalPosition = 2000, just as before, since this is the location that the user means when he types in 200. The first absolute move resulting is a fraction of a user unit shorter than the following moves. All the following moves will be exactly between user unit grid points.

A relative move of 100 user units from InternalPosition = 3 is seen as follows: The user believes he is in UserPosition = 0 and wishes to go to UserPosition = 100, so the move will end in InternalPosition = 1000.

To sum up:

Once power to the drive is on, a grid is created with 0 location where the motor was at that moment. Any position command will end at a point on that grid. If the motor is moved in any other way (manually, in jog mode or current mode), and ends up in a point that is not on the grid, the next position command will be a fraction of a user unit longer or shorter than requested.

Position change by the user:

If the user initiates a change to the position reading, the InternalPosition will be changed by the same distance as the UserPosition. For example:

InternalPosition = 953

UserPosition = 95

The user enters the command: UserPosition = 0. The user position is reduced by 95, which is equivalent to 950 encoder counts. The internal position is changed to InternalPosition = 3.

This is a must to prevent a drift in position reading by repeated position changes from the user, while not compromising the accuracy the user requested.

---

## Keywords Reference

---

## AmpFullScale

Description	Value
<b>Mnemonic (keyword):</b>	AmpFullScale
<b>CAN code:</b>	228
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	50,000
<b>Default value:</b>	5,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Abort

Abort will stop the current motion immediately. If Abort is entered when no motion is in progress it has no effect.

Description	Value
<b>Mnemonic (keyword):</b>	Abort
<b>CAN code:</b>	133
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Stop](#),



## About

---

This command is for internal use by the Agito PC suite only.

Description	Value
<b>Mnemonic (keyword):</b>	About
<b>CAN code:</b>	223
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

## AbsTrgt

AbsTrgt is the absolute target for a point to point and for repetitive motions. AbsTrgt defines the desired target position in the end of the motion regardless of the current position. For point to point motion this will be the location in which the motion will end. For repetitive motions AbsTrgt will be one of the locations where the motor stops, and the current position will be the other location.

If RelTrgt is not 0 than AbsTrgt is ignored and RelTrgt is used to determine the actual target of the next motion.

AbsTrgt can be changed during motion and the target of the current motion will change immediately.

Description	Value
<b>Mnemonic (keyword):</b>	AbsTrgt
<b>CAN code:</b>	134
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

### Example:

If the current location is Pos = 1000, AbsTrgt = 5000 and RelTrgt = 0, the next point to point motion will end when Pos = 5000.

If the current location is Pos = 1000, AbsTrgt = 5000 and RelTrgt = 7000, the next point to point motion will end when Pos = 8000 (Using RelTrgt and adding it to the current location).

If the current location is Pos = 1000, AbsTrgt = 5000 and RelTrgt = 0, the next repetitive motion will go to Pos = 5000 and then return to Pos = 1000 and so on.

### See Also:

[RelTrgt](#)

## Accel

Accel is the acceleration used for all motions in user units / sec<sup>2</sup>. This acceleration is also used with pulse – direction command input when the indirect mode is applied.

Accel can be changed during motion and the latest value will be effective immediately. If the current motion is still accelerating, the acceleration rate will be changed. If constant velocity was reached the new acceleration value will be used the next time acceleration is required.

Description	Value
<b>Mnemonic (keyword):</b>	Accel
<b>CAN code:</b>	136
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AbsTrgt](#), [RelTrgt](#), [Vel](#), [Speed](#), [Decel](#), [MaxAcc](#)

## AccelFact

Description	Value
<b>Mnemonic (keyword):</b>	AccelFact
<b>CAN code:</b>	168
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	40
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AccFFW

Agito controllers implement a position over velocity control filter scheme.

The position control filter consists of gain and several bi-quad filters. The actual number of the filters is product dependent. The velocity control filter consists of a PI filter and several (product dependent) bi-quad filters.

In addition, the control algorithm includes an acceleration feed-forward which is defined by the parameter AccFFW.

AccFFW[] is an array, to serve the gain scheduling mechanism. At any time, a given element of AccFFW[] is used for the control filter calculations, based on the current decision of the gain scheduling mechanism.

The default element is AccFFW[1].

The user may set all elements of AccFFW[] to be equal, so the same feed-forward gain value will be used with any decision of the gain scheduling mechanism. The user can set different values, so different feed-forward gains will be used for each case.

The relevant equations in the controller are (refer also to PosGain, VelGain and VelKI documentation, for the complete equations of the control filters):

$$\text{CurrRef} = \text{VelPIOutput} + \text{AccFFW}[\text{As Selected By Scheduling}] * (\text{PosRef} - 2 * \text{PosRefPrev} + \text{PosRefPrevPrev}) / 256$$

It is important to note that following these equations, CurrRef itself is limited by  $\pm \text{PeakCL}$  (PeakCL is a controller parameter).

Description	Value
<b>Mnemonic (keyword):</b>	AccFFW
<b>CAN code:</b>	101
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	50,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosGain](#), [VelGain](#), [VelKi](#), [PosFiltOn](#), [VelFiltOn](#), [PeakCL](#), [ScheduleMode](#).

## AccShapeDist

Description	Value
<b>Mnemonic (keyword):</b>	AccShapeDist
<b>CAN code:</b>	163
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AccShapeFact

Description	Value
<b>Mnemonic (keyword):</b>	AccShapeFact
<b>CAN code:</b>	164
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AccShapeOn

Description	Value
<b>Mnemonic (keyword):</b>	AccShapeOn
<b>CAN code:</b>	162
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

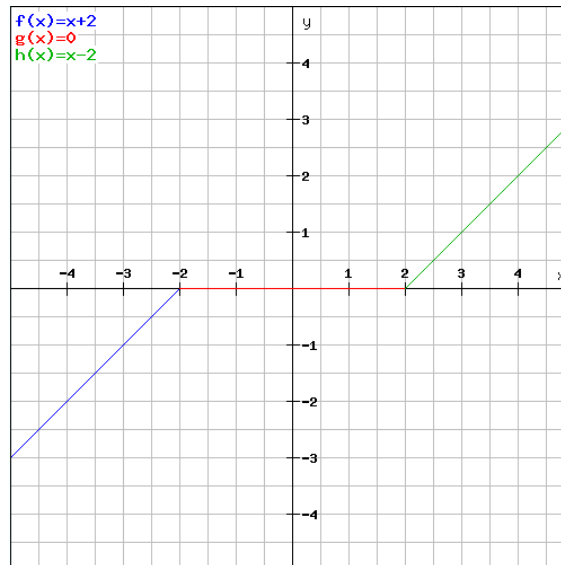


## AInDB

AInDB is the analog input dead band in milliVolts. If the actual input voltage is between (- AInDB) and AInDB the input reading will remain 0. Voltages above AInDB will be read as (actual voltage – AInDB). Values below (-AInDB) will be read as (actual voltage + AInDB).

The filter is calculated first. The offset is added to the input reading next. After adding offset the result is compared to the dead band (AInDB), the gain (AInGain) is applied to the result.

For example, if the dead band is 2 (no offset or gain), the voltage reading as a function of the input voltage will look like this:



Description	Value
Mnemonic (keyword):	AInDB
CAN code:	215
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Array with index range of:	1 : 4
Save to flash:	Yes
Axis related:	Yes
Min value:	0
Max value:	2,147,483,647
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

[AInOffset](#), [AInGain](#), [AInFilt](#), [AInPort](#)

## AlnFilt

AlnFilt is an array. Each location in the array corresponds with an analog input. The value that is entered to AlnFilt[1], for example, will determine the coefficients of the filter applied to input1. The coefficients are

$$C1 = \text{AlnFilt}[1]/65536 \text{ and } C2 = (1 - C1)$$

The filter is:

$$\text{Filtered Analog Input} = C1 * \text{Current input} + C2 * \text{Previous filtered value}$$

To have an unfiltered reading C1 should be 1, so AlnFilt[1] = 65536.

The filter is calculated first. The offset is added to the input reading next. After adding offset the result is compared to the dead band (AlnDB), the gain (AlnGain) is applied to the result.

Description	Value
<b>Mnemonic (keyword):</b>	AlnFilt
<b>CAN code:</b>	218
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	500,000
<b>Default value:</b>	10,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AlnOffset](#), [AlnDB](#), [AlnGain](#), [AlnPort](#)

## AlnGain

AlnGain is the gain of the analog input multiplied by 65536. The multiplication is needed to allow representation of fractions using only integers. For gain = 1 AlnGain = 65536.

The filter is calculated first. The offset is added to the input reading next. After adding offset the result is compared to the dead band (AlnDB), the gain (AlnGain) is applied to the result.

Description	Value
<b>Mnemonic (keyword):</b>	AlnGain
<b>CAN code:</b>	217
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	65,536
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[AlnOffset](#), [AlnDB](#), [AlnFilt](#), [AlnPort](#)

## AlnMode

Description	Value
<b>Mnemonic (keyword):</b>	AlnMode
<b>CAN code:</b>	257
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	4,128,776
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AlnOffset

AlnOffset determines the offset for the analog input in millivolt. The value of AlnOffset is added to the actual voltage on the input.

The filter is calculated first. The offset is added to the input reading next. After adding offset the result is compared to the dead band (AlnDB), the gain (AlnGain) is applied to the result.

Description	Value
<b>Mnemonic (keyword):</b>	AlnOffset
<b>CAN code:</b>	216
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[AlnDB](#), [AlnGain](#), [AlnFilt](#), [AlnPort](#)

## AlnPort

AlnPort is a read only array that contains the processed and original reading of the analog input port. The first half of the array holds the readings after they were filtered and after offset, dead band and gain were applied.

The second half of the array holds the original values of the input.

Description	Value
<b>Mnemonic (keyword):</b>	AlnPort
<b>CAN code:</b>	35
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[AlnFilt](#), [AlnDB](#), [AlnOffset](#), [AlnGain](#)

## AllStat

Using this non-axis-related keyword, the user can ask the controller to report a list of statuses (as selected by the user) for a group of axes (again, as selected by the user) as a respond to a single AllStat inquiry.

This keyword is supported over all communication connections. However, it is extremely efficient over the Ethernet connection, where a bulk of statuses can be reported within a single (or few, as needed) Ethernet packets.

AllStat keyword format follows the format of a function related, non-axis keyword such that

AllStat[ {AxisBitMask} ] , {GroupBitMask}

Where,

AxisBitMask indicates of which axis are being inquired. Such that bit 0 is A, bit 1 is B, bit 2 is C and so on. And that a value of 5 would mean that statuses from axis A and C are being referred to

Where,

GroupBitMask represents which group of statuses that will be returned in the function call such that a value of 10 would mean that statuses of groups 2 and 4 will be returned. There are a total of 7 groups that currently exist so the maximum number can be up to 127.

Please refer to AllStat file to get more internal information.

Description	Value
<b>Mnemonic (keyword):</b>	AllStat
<b>CAN code:</b>	420
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 63
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	511
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

AAllStat[3], 5

The "A" axis indication has no effect since the AllStat keyword is non-axis-related keyword.

"AllStat" is of course the keyword itself.

[3] ("array index") is a bitwise definition of the axes which are inquired. In this case, the value of 3 means axes A and B (right most bit – bit 0) is A axis, bit 1 is B axis and so on).

The value of 5 is used to select which group of statuses to report, again, using bitwise coding. Bit 0 (right most) refers to Group1 of statuses, bit 1 refers to Group2 and so on. Accordingly, a value of 5 refers to Group1 and Group3.

**See Also:**



## AmpType

Define the topology of the control system for axis. There are following several choices based on different amplifier type.

- 0 – Built in PWM amplifier (DRV product)
- 1 – Reserved. Please don't use it.
- 2 – Analog command to external driver.
- 3 – Pulse/Direction command to external driver.
- 4 – Built In linear amplifier. Please don't use it.

Description	Value
<b>Mnemonic (keyword):</b>	AmpType
<b>CAN code:</b>	226
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	4
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

### See Also:

AmpFullScale, LAmpFullScale, PowerSupply.

## AntiCoggAmp

Description	Value
<b>Mnemonic (keyword):</b>	AntiCoggAmp
<b>CAN code:</b>	408
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	5,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AntiCoggOn

Description	Value
<b>Mnemonic (keyword):</b>	AntiCoggOn
<b>CAN code:</b>	407
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AntiCogPhase

Description	Value
<b>Mnemonic (keyword):</b>	AntiCogPhase
<b>CAN code:</b>	409
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	359
<b>Default value:</b>	180
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AntiCoggValue

Description	Value
<b>Mnemonic (keyword):</b>	AntiCoggValue
<b>CAN code:</b>	410
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AOutMode

AOutMode determines which parameters will be monitored by the analog outputs.

AOutMode[1] determines which parameter will be reflected by Analog Out 1, AOutMode[2] determines which parameter will be reflected by Analog Out 2 and so on.

The value of AOutMode is in "Complex CAN code". This means that it includes the CAN code of the requested parameter + other information necessary to determine what will be monitored by the analog output.

The complex CAN code is a 32 bit variable. The 32 bits are divided into the following bit fields:

Bits 0 – 9 (LSB) include the CAN code of the parameter in 10 bits.

Bits 10 – 12 are axis number in 3 bits. 0 is the first axis.

Bits 16 – 31 are the index into an array for arrays (ignored otherwise).

The value of the monitored parameter is sent to the analog output and is considered to be in mV. So if PosErr is monitored and it is equal to 120, the analog output will be 120mV.

Description	Value
<b>Mnemonic (keyword):</b>	AOutMode
<b>CAN code:</b>	220
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

To monitor Vel[2] on analog output 1:

First we should calculate the complex CAN code:

The CAN code for Vel is 5.

For a single axis controller (or for a multi axis controller, axis A), the axis number is 0.

The index into the array is 2.

The hexadecimal word is 0x00020005

In decimal: 131,077

So, we should enter AOutMode[1] = 131077

### See Also:

[AOutPort](#), [AOutShifts](#)

## AOutOffset

Description	Value
<b>Mnemonic (keyword):</b>	AOutOffset
<b>CAN code:</b>	227
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-500
<b>Max value:</b>	500
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AOutPort

The value entered into AOutPort[1] will appear on Analog Out 1 if AOutMode[1] = 0. The value entered into AOutPort[2] will appear on Analog Out 2 if AOutMode[2] = 0.  
The value of AOutPort is in mV. Please refer to the hardware manual for the DAC resolution and output Voltage.

Description	Value
<b>Mnemonic (keyword):</b>	AOutPort
<b>CAN code:</b>	219
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-12,000
<b>Max value:</b>	12,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[AOutMode](#), [AOutShifts](#)



## AOutShifts

AOutShifts divides or multiplies the monitored parameter (refer to AoutMode) to obtain a better dynamic range. AOutShifts[1] is applied to the value on Analog Output 1 and AOutShifts[2] is applied to the value on Analog Output 2.

A negative value in AOutShifts will cause a shift right by the value of AOutShifts. This is the same as dividing the output value by  $2^{\text{abs}(\text{AOutShifts})}$ . A positive value of AOutShifts will cause a shift left by the value of AOutShifts. This is the same as multiplying the output value by  $2^{\text{AOutShifts}}$ .

Analog output in mV = [Value of the monitored parameter in internal units] \*  $2^{\text{AOutShifts}}$

Description	Value
<b>Mnemonic (keyword):</b>	AOutShifts
<b>CAN code:</b>	221
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-31
<b>Max value:</b>	31
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

Suppose that Analog Output 2 displays (monitors) PosErr – the position error. We expect, in this example, PosErr to be a value of no more than 10. Without using AOutShifts we will receive output values of no more than 10mV. This could be inconvenient, especially if there is noise on the output. To use more of the output range we can multiply it by any number we want up to D/A range limitation. For example, we will choose 256, so AOutShifts[2] = 8 (Positive for multiplication).

### See Also:

[AoutMode\[\]](#) and [AOutPort\[\]](#).

## AutoExec

AutoExec = 1 will cause the user program to start executing automatically on power up or after software restart. Obviously "Save" command must be used before reset to save the value of AutoExec to flash memory.

Description	Value
<b>Mnemonic (keyword):</b>	AutoExec
<b>CAN code:</b>	208
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGAccTh

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGAccTh
<b>CAN code:</b>	353
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGBW

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGBW
<b>CAN code:</b>	358
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,000
<b>Default value:</b>	20
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**  
autogain

## AutoGCopy

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGCopy
<b>CAN code:</b>	350
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGDownSm

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGDownSm
<b>CAN code:</b>	359
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	6
<b>Default value:</b>	4
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGJm

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGJm
<b>CAN code:</b>	351
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	3,993
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGJratUs

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGJratUs
<b>CAN code:</b>	363
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-50
<b>Max value:</b>	20,000
<b>Default value:</b>	140
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

Autogain,



## AutoGKt

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGKt
<b>CAN code:</b>	362
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	38,231
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGMask

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGMask
<b>CAN code:</b>	370
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGMaxLen

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGMaxLen
<b>CAN code:</b>	356
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	20
<b>Max value:</b>	100
<b>Default value:</b>	30
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGMaxRat

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGMaxRat
<b>CAN code:</b>	365
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	20,000
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGMinLen

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGMinLen
<b>CAN code:</b>	355
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	10
<b>Max value:</b>	100
<b>Default value:</b>	15
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGMinRat

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGMinRat
<b>CAN code:</b>	364
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-40
<b>Max value:</b>	20,000
<b>Default value:</b>	-10
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGMode

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGMode
<b>CAN code:</b>	367
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	5
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGNumSet

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGNumSet
<b>CAN code:</b>	369
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	5
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## AutoGOn

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGOn
<b>CAN code:</b>	361
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGPosFilt

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGPosFilt
<b>CAN code:</b>	354
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	1,000
<b>Default value:</b>	50
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGQualTh

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGQualTh
<b>CAN code:</b>	368
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	1,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGSavPer

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGSavPer
<b>CAN code:</b>	366
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	16
<b>Default value:</b>	5
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGStatus

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGStatus
<b>CAN code:</b>	357
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 50
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AutoGVelTh

This keyword is part of the autogain function. For details about autogain and how it works please read the autogain chapter in the user's manual.

Description	Value
<b>Mnemonic (keyword):</b>	AutoGVelTh
<b>CAN code:</b>	352
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	5,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AuxEncDir

Enables inversed reading of the auxiliary encoder direction (for all types of encoders).

If AuxEncDir == 0, the auxiliary encoder is read as connected to the drive.

If AuxEncDir == 1, the auxiliary encoder is read with inversed direction.

Description	Value
<b>Mnemonic (keyword):</b>	AuxEncDir
<b>CAN code:</b>	61
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## AuxEncFilt

Specifies the digital filter to apply on each of the incremental auxiliary encoder input channels (A, B and Index).

The filter is implemented by hardware.

The filter is characterized as follows:

- The input is sampled by the filter mechanism. An input level is qualified only after 6 consecutive samples with identical value. This means that the "1" logic of the signal needs to be sampled at least 6 times and similarly also the "0" logic, to a total of  $(2 * 6)$  samples.
- The filter frequency (sampling frequency) is determined by the AuxEncFilt parameter.
- If AuxEncFilt == 0, the filter frequency is equal to the DSP clock. Typically with Agito products: 300MHz.
- If AuxEncFilt is not equal to 0, the filter frequency is:

$$\text{Filter Frequency} = 300 \text{ MHz} / (2 * \text{AuxEncFilt})$$

- The maximal theoretical input frequency, at A and B and Index signals (when no noises are presented) is therefore calculated using the following equation:

If AuxEncFilt = 0

$$\text{Max Input Frequency} = 300 \text{ MHz} / (2 * 6) \quad \text{MHz}$$

Else

$$\text{Max Input Frequency} = 300 \text{ MHz} / (\text{AuxEncFilt} * 2) / (2 * 6) \quad \text{MHz}$$

Notes:

- The "Max Input Frequency" is the theoretical upper limit of the input frequency as a function of a given EncFilt value. It is calculated assuming ideal square wave signal (infinite slew rate) and ideal electronics (no delays and slew rate at the receiver chips in the controller). The actual maximal frequency that can be used will be somehow smaller, depending on the quality of the signals.
- In addition, if noises are presented in the signal, it will "disturb" the filter from counting 6 consecutive samples of a given logic level. So, again, it is recommended not to push AuxEncFilt too high.
- Bottom line, with a given Max Input Frequency, we recommend setting EncFilt as follows:

$$\text{AuxEncFilt} \leq 300 / 24 / \text{Max Input Frequency (MHz)} / 2$$

$\leq$  means smaller or equal.

The division by 2 is to provide the spares required for none optimal signals and noises.

- The actual value to set for AuxEncFilt at a given system can be initially set according to the above equation but must be carefully tested and adjusted at the system as suitable for the noises that appears on the encoder signals.



- The fastest input signal can be (from the filter point of view): 25MHz (with AuxEncFilt = 0 and assuming no noises).
- The fastest filter frequency is 300 MHz (EncFilt = 0) and the slowest filter frequency is ~0.59 MHz (maximal value for EncFilt is 255).

Description	Value
<b>Mnemonic (keyword):</b>	AuxEncFilt
<b>CAN code:</b>	60
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	255
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EncFilt](#), [AuxEncType](#), [AuxEncSubType](#),

## AuxEncSubType

Description	Value
<b>Mnemonic (keyword):</b>	AuxEncSubType
<b>CAN code:</b>	611
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxEncType](#),

## AuxEncType

AuxEncType sets the auxiliary encoder feedback type. The values that can be assigned to AuxEncType are listed below. Note that some of these encoder types are not available on all products and some require special options.

Value	Encoder Type
0	Unknown
1	Incremental encoder
2	Sin/Cos
3	Endat
4	SSI
5	Nikon 17 bit encoder

Description	Value
<b>Mnemonic (keyword):</b>	AuxEncType
<b>CAN code:</b>	59
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	6
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxEncSubType](#),

## AuxIndexPos

The AuxIndexPos holds the last auxiliary encoder position at which an auxiliary encoder index was detected.

The AuxIndexPos works only with incremental encoder.

Note that to properly detect an index signal and its position, the encoder velocity must be smaller than the controller sampling frequency, so that the index pulse width will be at least as wide as a single sampling.

Description	Value
<b>Mnemonic (keyword):</b>	AuxIndexPos
<b>CAN code:</b>	48
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	Aux user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxIndexStat](#), [IndexStat](#) and [IndexPos](#).

## AuxIndexStat

The AuxIndexStat holds the current status ("0" or "1") of the auxiliary encoder index input (for incremental encoder only).

Description	Value
<b>Mnemonic (keyword):</b>	AuxIndexStat
<b>CAN code:</b>	46
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxIndexPos](#), [IndexStat](#) and [IndexPos](#).

## AuxLockCntr

Auxiliary lock counter reports the number of times a change in the input designated as auxiliary lock source was detected since the latest enabling of auxiliary lock. The value is reset by AuxLockEn = 0, AuxLockEn = 1. It can also be reset by assigning 0.

Description	Value
<b>Mnemonic (keyword):</b>	AuxLockCntr
<b>CAN code:</b>	176
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxLockEn](#), [AuxLockVal](#), [AuxLockCntr](#), [AuxLockSrc](#)

## AuxLockEn

AuxLockEn = 1 will enable the position lock on the auxiliary feedback. When position lock is enabled a change in the designated input (rise for CW motion, fall for CCW) will lock the auxiliary encoder position.

The value of the position during the input change is saved in "AuxLockVal". The number of times a position was locked is saved in "AuxLockCntr".

For incremental encoder, the position is locked by hardware, so it is very accurate. For absolute encoder the position that is read in the next sampling after the lock input changed is saved in "AuxLockVal".

AuxLockVal and AuxLockCntr are updated only once every sample time. If more than one valid change in the input happens during a single sample time only the last change will be recorded.

Description	Value
<b>Mnemonic (keyword):</b>	AuxLockEn
<b>CAN code:</b>	174
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxLockVal](#), [AuxLockCntr](#), [AuxLockSrc](#)

## AuxLockSrc

AuxLockSrc is the number of input used as source for the auxiliary lock feature. Changes on this designated input will be reflected in AuxLockVal and AuxLockCntr. The same input that is designated as auxiliary lock source can also have a different role assigned to it in DInMode.

Description	Value
<b>Mnemonic (keyword):</b>	AuxLockSrc
<b>CAN code:</b>	175
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-32
<b>Max value:</b>	32
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxLockEn](#), [AuxLockVal](#), [AuxLockCntr](#), [DInMode](#)



## AuxLockVal

AuxLockVal holds the value of the latest auxiliary position where the lock input changed. This value is saved by hardware for an incremental encoder. For absolute encoder the value of the position during the next sample is saved.

Description	Value
<b>Mnemonic (keyword):</b>	AuxLockVal
<b>CAN code:</b>	177
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	Aux user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxLockEn](#), [AuxLockSrc](#), [AuxLockCntr](#)

## AuxPos

AuxPos reports the auxiliary encoder position reading in auxiliary user units (AuxUsrUnits). If AuxUsrUnits = 1 then AuxPos is in encoder counts. The value of AuxPos is 0 upon reset. AuxPos counts the auxiliary position between -2147483648 cnts and 2147483647. If the actual position reading exceeds these limits the position reading will roll. To prevent this from happening use the modulo function (AuxModRev).

With the motor off AuxPos can also be set by the user to a desired value.

Description	Value
<b>Mnemonic (keyword):</b>	AuxPos
<b>CAN code:</b>	3
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	Aux user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AuxUsrUnits](#), [AuxModRev](#)

## AuxUsrUnits

AuxUsrUnits allows the user to read the auxiliary feedback position and its derivatives in units other than encoder counts. AuxUsrUnits is the ratio between the desired unit and the encoder counts. A full explanation of how user units work can be found in the beginning of this document.

Description	Value
<b>Mnemonic (keyword):</b>	AuxUsrUnits
<b>CAN code:</b>	65
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	65,536
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If the user wants to see the auxiliary position reading in mm, and every 5 encoder counts on the auxiliary encoder are equivalent to 1mm, set UsrUnits to 5.

The auxiliary position reading will now be received in mm, velocity in mm/sec and acceleration in mm/sec<sup>2</sup>.

### See Also:

[UsrUnits](#), [PUsrUnits](#)

## AuxVel

AuxVel reports the velocity of the auxiliary encoder in auxiliary user units. The auxiliary velocity is reported in user-units / second

Description	Value
<b>Mnemonic (keyword):</b>	AuxVel
<b>CAN code:</b>	6
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	Aux user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Begin

Briefly:

The "Begin" message initiates a motion (assuming MotorOn is 1, i.e.: enabled) according to the value of MotionMode and other related parameters (such: as acceleration, speed and target position).

The Begin command is used to begin a motion after all its parameters were set. This command will take effect only if the motor is enabled. Otherwise, an error will occur.

The type of motion that will begin depends on MotionMode.

The motion that begins uses all the relevant parameters as they were set before the Begin command was entered: speed, acceleration, deceleration, jerk, and position target.

After "Begin" is entered the motion parameters can be changed on the fly and will take effect immediately.

Description	Value
<b>Mnemonic (keyword):</b>	Begin
<b>CAN code:</b>	131
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[MotionMode](#),

## BeginDInOn

Using the parameter BeginDInOn (Begin Digital Input On), the user can define that a motion (working with any of the available Motion Modes) will not actually start when the Begin command is sent to the controller but only after a rising edge is detected at a specified (user-defined) discrete input (any of the optically isolated, or differential or Sync inputs).

Using this feature, a motion can be synchronized with an external event. By using the same signal as an input to multiple numbers of controllers, a multi-axes synchronized motion can be created.

When BeginDInOn == 0, this feature is disabled and motions will start immediately following the Begin command.

When BeginDInOn == 1, a Begin command is still required to initiate a motion, but the motion will not actually start until a rising edge will be detected at a user-specified digital input.

Notes (when BeginDInOn == 1):

The value of the BeginDInOn parameter is monitored only when Begin is received by the controller. If it is 0, a standard Begin Motion is executed. If it is 1, the motion is flagged, but is suspended till the user defined input has a rising edge (from "0" to "1").

Note that DInLog can be used to invert this logic, as suitable for your system.

Otherwise than during the Begin processing, the value of BeginDInOn is ignored. Changing it will not have any effect during a motion or during the time of waiting for the input to rise.

Upon Begin command, if BeginDInOn == 1, the MotionStat is flagged, as in typical motion that the axis is in motion (bit 0). However, bit 9 is also set, to flag that the motion is suspended till the rising edge at the selected digital input.

Refer to the MotionStat keyword page for details about it various bits.

While waiting for the input to rise, a Stop command will terminate the motion and the waiting process.

The MotionSamples[] counts the time of the motion. Note that the waiting time is not counted and the MotionSamples[] starts to count only following the rising edge, when the motion is actually starting.

Note that the motion is actually starting one control sample (typically with Agito controllers: ~61(s) following the rising edge at the selected input.

Note that the "accuracy" (or jitter) of the rising edge detection is one control sample. So, practically, the motion for a given controller will start 61(s - 122(s following the rising edge.

The input that will be used to start the motion is selected using the DInMode[] parameter. If DInMode[n] == 3, the n input will be used. Refer to the DInMode[] keyword page for detailed description.

Following the rising edge, the input must stay at its active level for at least one sample time, to ensure that its value is sampled and detected by the software (that samples the digital inputs each control sample).

Any of the controller input can be used. This includes the optically isolated inputs, differential inputs and the Sync input, if supported at the specific product.

Some of the motions (like ECAM, Gearing, and Pulse/Direction) performs motions that are relative to the input value of a master (or pulses counter) at the time of the Begin. When using the Begin-On-Input feature, the user must take care that the initial condition of the relevant master will not change between the time of the Begin and the time of the rising edge at the input. Otherwise, unexpected motion can happen when at the time of the rising edge.

Description	Value
<b>Mnemonic (keyword):</b>	BeginDInOn
<b>CAN code:</b>	142
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Begin](#), [MotionStat](#), [Stop](#) and [DInMode](#).

## BeginOnToPos

Description	Value
<b>Mnemonic (keyword):</b>	BeginOnToPos
<b>CAN code:</b>	587
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## BrakeLockTime

Description	Value
<b>Mnemonic (keyword):</b>	BrakeLockTime
<b>CAN code:</b>	381
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	10
<b>Max value:</b>	799
<b>Default value:</b>	99
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## BrakeMode

Description	Value
<b>Mnemonic (keyword):</b>	BrakeMode
<b>CAN code:</b>	380
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	4
<b>Default value:</b>	2
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## BrakeRelTime

Description	Value
<b>Mnemonic (keyword):</b>	BrakeRelTime
<b>CAN code:</b>	382
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	10
<b>Max value:</b>	799
<b>Default value:</b>	99
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## BrakeUsed

If BrakeUsed = 0 the static brake will not be released and PC Suite will not display the static brake configuration fields. This parameter will not change the other brake related parameters. This is for the dedicated output or for a general purpose output that has the relevant mode.

Description	Value
<b>Mnemonic (keyword):</b>	BrakeUsed
<b>CAN code:</b>	379
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	1
<b>Implementation status:</b>	Firmware: 1.2.0

**Example:**

**See Also:**

## CalcFilters

Agito controllers include some bi-quad filters within the position and velocity control loops. The coefficients of these filters are saved at the VelFilt[] and PosFilt[] array parameters. Refer to the description of VelFilt[] and PosFilt[] for the detailed structure of these filters. However, the user does not write directly to VelFilt[] and PosFilt[]. Instead, the user uses the VelFiltDef[] and PosFiltDef[] (as well as: VelFiltOn[], PosFiltOn[]) to easily define the various filters based on their characteristics (in contrast to by their coefficients). Once the user enters new values to (at least) one of: VelFiltOn[], VelFiltDef[], PosFiltOn[] and PosFiltDef[], the user needs to execute CalcFilters to calculate and to properly updated the VelFilt[] and PosFilt[] parameters. Upon successful execution of CalcFilters, the newly defined filters become active.

### Notes:

Once the user changes (at least) one of: VelFiltOn[], VelFiltDef[], PosFiltOn[] and PosFiltDef[], it will be impossible to enable the drive until CalcFilters is successfully executed. This is to ensure that the closed loop will use the recent filters definitions. CalcFilters may fail if the input characteristics of the filters (at VelFiltDef[] or PosFiltDef[]) are wrong or out of range. In such case, it will return with a proper error code. Again, it will not be possible to enable the drive till the problem is fixed and CalcFilters is executed successfully. CalcFilters calculates both the velocity filters and the positions filters.

Refer to the detailed description of VelFiltOn[], VelFiltDef[], PosFiltOn[] and PosFiltDef[] for description of the supported filters and their definitions.

Description	Value
<b>Mnemonic (keyword):</b>	CalcFilters
<b>CAN code:</b>	360
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

### See Also:

## CalcIden

Identification is the process of measuring the system frequency response.

With Agito controllers and PC Suite, the identification process is managed by the PC Suite. It involves injection of user defined sinus commands and measurement of the response for each frequency.

However, once the experiment with a given single frequency is completed, it is the drive (with the properly recorded data of injected signal and output signal) that calculates the system response at this frequency.

This calculation is performed using the CalcIden function keyword.

The CalcIden function assumes that the drive holds a recorded data with two vectors. The first is the input signal and the second is the output signal. It also assumes that the InjectFreq parameter holds the input frequency.

It uses this data to calculate the value of the transfer function at this frequency, based on the recorded data.

The calculated values are:

Gain [dB]

Phase [degrees]

Noise quality [%]

2nd harmonic quality [%].

This data is stored at the IdenResults[] array parameter and it is read by the PC Suite as part of the identification process (which repeatedly performs this process per each requested frequency).

The details of the data at IdenResults[] and the assumptions of CalcIden are not important for the user as the overall process is managed by the PC Suite. Please advise Agito in case you need additional information.

Description	Value
<b>Mnemonic (keyword):</b>	CalcIden
<b>CAN code:</b>	128
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CANAddr

Description	Value
<b>Mnemonic (keyword):</b>	CANAddr
<b>CAN code:</b>	67
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	2,032
<b>Default value:</b>	64
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CANBaud

The CANBaud parameter defines the baudrate of the CAN communication channel.

The following table describes the various supported CAN baud rates and the related CANBaud values:

CANBaud value	CAN Bus Communication Baudrate [KHz]
1	31.25
2	62.5
3	125
4	250
5	500
6	1,000
Other	1,000

Description	Value
<b>Mnemonic (keyword):</b>	CANBaud
<b>CAN code:</b>	68
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	6
<b>Default value:</b>	6
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## CANDelay

Some CAN devices can support a limited rate of messages. If such a device is used on the network some delay must be introduced. CANDelay introduces a delay between any two messages that are sent in CAN. The delay is in controller sample time units (CANDelay = 2 waits 2 sample times between messages).

To use the maximum available CAN speed set CANDelay to 0.

Description	Value
<b>Mnemonic (keyword):</b>	CANDelay
<b>CAN code:</b>	222
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	6
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ChainAddress

Description	Value
<b>Mnemonic (keyword):</b>	ChainAddress
<b>CAN code:</b>	159
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-1
<b>Max value:</b>	8
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIAutoConnect

Description	Value
<b>Mnemonic (keyword):</b>	CIAutoConnect
<b>CAN code:</b>	500
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIConnect

Description	Value
<b>Mnemonic (keyword):</b>	CIConnect
<b>CAN code:</b>	504
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIDeviceType

Description	Value
<b>Mnemonic (keyword):</b>	CIDeviceType
<b>CAN code:</b>	503
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	65,537
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIDisconnect

Description	Value
<b>Mnemonic (keyword):</b>	CIDisconnect
<b>CAN code:</b>	505
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIGlobalStat

Description	Value
<b>Mnemonic (keyword):</b>	CIGlobalStat
<b>CAN code:</b>	510
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIIdentity

Description	Value
<b>Mnemonic (keyword):</b>	CIIdentity
<b>CAN code:</b>	509
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 14
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## CIOfflineData

Description	Value
<b>Mnemonic (keyword):</b>	CIOfflineData
<b>CAN code:</b>	501
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIOfflineDef

Description	Value
<b>Mnemonic (keyword):</b>	CIOfflineDef
<b>CAN code:</b>	507
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CiOfflineSend

Description	Value
<b>Mnemonic (keyword):</b>	CiOfflineSend
<b>CAN code:</b>	502
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CIStatus

Description	Value
<b>Mnemonic (keyword):</b>	CIStatus
<b>CAN code:</b>	508
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 7
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CISyncDef

Description	Value
<b>Mnemonic (keyword):</b>	CISyncDef
<b>CAN code:</b>	506
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ClearErr

The ClearErr command clears ErrLog array.

Description	Value
<b>Mnemonic (keyword):</b>	ClearErr
<b>CAN code:</b>	236
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ErrLog](#),

## ComtAng

---

ComtAng is the commutation angle in encoder counts, or actually the distance in encoder counts between the current position and the actual electrical zero position of the motor. The commutation angle is the angle of the motor within the current electric cycle.

Example:

Assume a 2 pole pairs motor with 4000 cnts encoder. Each electric cycle is 2000 counts. If the motor position is currently 500 counts from the location detected as the electrical zero then ComtAng = 500, and this means that the motor is 90 degrees from the electrical zero.

Note:

Electrical zero position is defined as the motor location where the sinusoidal BEMF (or Torque/Force) "constant" of motor's phase A is 0.

## CNCAbsTrgt

The CNCAbsTrgt is, for example, the distance to move, at the currently active segment, along the CNC path.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAbsTrgt
<b>CAN code:</b>	464
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[CNCAccel](#), [CNCSpeed](#), [CNCADecel](#), [CNCAEndSpeed](#), [CNCAPosRef](#), [CNCAPosRef](#),



## CNCAAccel

Description	Value
<b>Mnemonic (keyword):</b>	CNCAAccel
<b>CAN code:</b>	458
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAClear](#), [CNCAdPosRef](#), [CNCASpeed](#), [CNCADecel](#), [CNCAEndSpeed](#), [CNCAAbsTrgt](#), [CNCAPosRef](#),

## CNCAClear

Description	Value
<b>Mnemonic (keyword):</b>	CNCAClear
<b>CAN code:</b>	455
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[CNCAdPosRef](#), [CNCASpeed](#), [CNCADecel](#), [CNCAEndSpeed](#), [CNCAAbsTrgt](#), [CNCAPosRef](#),

## CNCADecel

Description	Value
<b>Mnemonic (keyword):</b>	CNCADecel
<b>CAN code:</b>	459
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAAccel](#), [CNCASpeed](#), [CNCAAdPoRef](#), [CNCAEndSpeed](#), [CNCAAbsTrgt](#), [CNCAPosRef](#),

## CNCADoStep

The value of CNCADoStep can be written at any time, even during motion.

If CNCA motion is active, and if the CNC is in step mode (CNCAStepMode=1, see above), then setting CNCADoStep to 1 will instruct the controller to continue to the next step.

The value of CNCADoStep has no effect beside the conditional effect as described above.

Once CNCADoStep is set to 1, and once the controller reacts to this request and move to the next segment (only after reaching the end of the current segment), the controller clears CNCADoStep to 0, to ensure it will not perform more than a single segment.

Upon beginning a CNCA motion (using the Begin message), the value of CNCADoStep is automatically set as follows:

- If CNCAStepMode is 0, CNCADoStep is cleared to 0 as well (to be ready for activating the step mode during the motion).
- If CNCAStepMode is 1, CNCADoStep is set to 1, to ensure the first segment is executed.

Description	Value
<b>Mnemonic (keyword):</b>	CNCADoStep
<b>CAN code:</b>	471
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CNCAdPosRef

CNCAdPosRef is the derivative of CNCAPosRef, meaning the current vector velocity of the profiler.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAdPosRef
<b>CAN code:</b>	467
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAAccel](#), [CNCA Speed](#), [CNCA Decel](#), [CNCA EndSpeed](#), [CNCA AbsTrgt](#), [CNCA PosRef](#),

## CNCAEmrgDec

The CNC vector deceleration to use in case of emergency stop (hitting hardware Reverse /Forward Limit Switches, or reaching software RevPLim/FwdPLim position limits).

Description	Value
<b>Mnemonic (keyword):</b>	CNCAEmrgDec
<b>CAN code:</b>	461
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAAccel](#), [CNCASpeed](#), [CNCADecel](#), [CNCAEndSpeed](#), [CNCAAbsTrgt](#), [CNCAPosRef](#), [CNCAAdPosRef](#),

## CNCAEncRatio

This parameter is axis related and it should be separately set for each member axis.

The parameter defines the ratio between the resolutions of an axis to the other axes, to allow accurate CNC calculations for non-identical physical resolutions of the member axes.

This is a future feature (the controller supports this parameter, but it has no effect on the CNC calculations).

Currently, the CNC motion calculations assume identical resolution for all member axes.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAEncRatio
<b>CAN code:</b>	468
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	256
<b>Max value:</b>	25,600
<b>Default value:</b>	256
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAJerk](#),

## CNCAEndSpeed

Description	Value
<b>Mnemonic (keyword):</b>	CNCAEndSpeed
<b>CAN code:</b>	463
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[CNCAAccel](#), [CNCASpeed](#), [CNCADecel](#), [CNCAAbsTrgt](#), [CNCAPosRef](#), [CNCAAdPosRef](#)



## CNCAFIFO

Description	Value
<b>Mnemonic (keyword):</b>	CNCAFIFO
<b>CAN code:</b>	450
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 11000
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CNCAJerk

The parameter defines the Jerk (smoothing) to use when calculating the CNC vector motion profile. This is a future feature (the controller supports this parameter, but it has no effect on the CNC calculations).

Currently, the CNC vector motion profiler is calculated without smoothing.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAJerk
<b>CAN code:</b>	460
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	9
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[CNCAEncRatio](#),

## CNCAPause

When set to "1", the CNC motion decelerate to zero vector velocity.

When set to "0", the CNC motion is performed normally. If it was paused, it accelerates back to the desired vector speed of the active segment.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAPause
<b>CAN code:</b>	465
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CNCAPercents

Used by the user to scale the CNC speed (and acceleration/deceleration) along the CNC path. By affecting the speed and acceleration/deceleration, the CNCAPercents actually affects the duration of time that will be needed to perform the CNC motion.

A value of 100 (%) means that the motion will be according to the values defined in the CNC FIFO segments. A value of 50 (%), for example, means that the CNC will be performed at twice the time that was needed to perform the nominal CNC motion as defined in the CNC FIFO.

CNCAPercents can get values higher than 100 (%).

CNCAPercents can be modified at any time, including on-the-fly during the CNC motion.

User need to consider what value to give to CNCAPercents before starting a CNC motion.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAPercents
<b>CAN code:</b>	462
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	200
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CNCAPosRef

The CNCAPosRef is the current desired position along the CNC path (starting from zero and reaching CNCAAbsTrgt at the end of the segment).

Description	Value
<b>Mnemonic (keyword):</b>	CNCAPosRef
<b>CAN code:</b>	466
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAAccel](#), [CNCASpeed](#), [CNCADecel](#), [CNCAEndSpeed](#), [CNCAAbsTrgt](#), [CNCAAdPosRef](#),

## CNCAPushParam

Used to push values of parameters into the CNC FIFO.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAPushParam
<b>CAN code:</b>	453
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAPushSeg](#), [CNCAPushType](#),

## CNCAPushSeg

This keyword is supported only over Ethernet communication connection to the controller. It will return an error if used over any other communication connection (such as RS232, CAN, etc.). Using this keyword, the host/user can push a complete segment (type and parameters) using a single Ethernet message.

This replaces a sequence of messages consisting me CNCAPushType followed by multiple CNCAPushParam as required for the segment type.

As a result, it significantly improves the throughput of pushing segments into the CNC FIFO.

Description	Value
<b>Mnemonic (keyword):</b>	CNCAPushSeg
<b>CAN code:</b>	469
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAPushType](#), [CNCAPushParam](#),

## CNCAPushType

Description	Value
<b>Mnemonic (keyword):</b>	CNCAPushType
<b>CAN code:</b>	452
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	16,777,216
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAPushParam](#), [CNCAPushSeg](#),



## CNCARemove

A function keyword to remove the last CNC FIFO A segment.

If the last pushed segment is closed, it will be popped out from the CNC FIFO (removed from the CNC FIFO) and its ID will be also cancelled (so next pushed segment will get this ID).

If we are in the middle of pushing parameters to a segment, this segment will be cancelled, and the controller will be ready to get a new segment (which will get the same ID as the deleted segment, of course).

CNCARemove will return an error if the CNC FIFO A is empty.

CNCARemove will fail and return an error (without removing anything from the CNC FIFO) if the controller is using this segment for motion at this time.

Description	Value
<b>Mnemonic (keyword):</b>	CNCARemove
<b>CAN code:</b>	454
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CNCASpeed

CNCASpeed represent the desired vector speed along the CNC path, for the currently active segment. However, the actual vector speed is this value multiplied by speed factor defined as part of the CNC FIFO definition and by a second factor defined on-the-fly by the user (CNCAPercents keyword).

Description	Value
<b>Mnemonic (keyword):</b>	CNCASpeed
<b>CAN code:</b>	457
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CNCAAccel](#), [CNCAdPosRef](#), [CNCADecel](#), [CNCAEndSpeed](#), [CNCAAbsTrgt](#), [CNCAPosRef](#),

## CNCASatus

A read only array parameter holding the following described status data.

Some of the statuses are not valid after reset or power on, some others are not valid when the CNC FIFO is empty and some other when there is no CNC FIFO motion and so on. These statuses get the value -1 when they are not valid (this value is not a valid value for any of these statuses when they are valid).

Please refer to [Motion Modes-CNC](#) for more detailed information.

Description	Value
<b>Mnemonic (keyword):</b>	CNCASatus
<b>CAN code:</b>	451
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CNCASStepMode

The value of CNCASStepMode can be written at any time, even during motion.

When 0, the CNC motion will act normally (no step mode).

When 1, the CNC engine will halt at the end of each segment and will wait for CNCADoStep = 1 (see below) to perform the next segment (and halt again at the end of that segment).

The End Speed of each segment is forced to 0, even if a different value is defined as part of the segment definition.

Any user command to stop the motion (StopCNCA, Stop, and Abort) will force CNCASStepMode parameter to 0.

Note that CNCASStepMode can be modified while the controller is in motion. So, user can enter the step mode at any time (the controller will halt at the end of the currently executed segment) by setting this parameter to 1. On the other hand, user can leave the step mode and the controller will continue to freely execute the NC segments by writing 0 to this parameter.

Description	Value
<b>Mnemonic (keyword):</b>	CNCASStepMode
<b>CAN code:</b>	470
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Compare

Compare is a user program low level language keyword. The syntax related to “Compare” is usually generated automatically by the PC suite during compilation.

Compare pops two variables from the numeric stack and compares them as required. If the result is “true” it will push “1” into the stack. If the result is “false” it will push “0”.

The index value determines which operation will be performed. The operands should be pushed to the stack before Compare is called.

The result is pushed to the numeric stack. If this function is called from communication the value is also sent through communication.

The values of “Compare” (pop1 is the top value in the stack, pop2 is the next value):

Value	Operation type
2	“==” true if pop1 == pop2
3	“>” true if pop1 > pop2
4	“>=” true if pop1 >= pop2
5	“<” true if pop1 < pop2
6	“<=” true if pop1 <= pop2
7	“!=” true if pop1 != pop2
8	“Zero” true if pop1 == 0
9	“ Not Zero” true if pop1 !=0

(The values start with 2 for compatibility with other functions)

Description	Value
<b>Mnemonic (keyword):</b>	Compare
<b>CAN code:</b>	195
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 9
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ComtAng

ComtAng is the commutation angle in encoder counts, or actually the distance in encoder counts between the current position and the actual electrical zero position of the motor. The commutation angle is the angle of the motor within the current electric cycle.

Description	Value
<b>Mnemonic (keyword):</b>	ComtAng
<b>CAN code:</b>	73
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

Assume a 2 pole pairs motor with 4000 cnts encoder. Each electric cycle is 2000 counts. If the motor position is currently 500 counts from the location detected as the electrical zero then ComtAng = 500, and this means that the motor is 90 degrees from the electrical zero.

### See Also:

## ComtMode

Commutation is the process of alternating the current between the different motor phases to generate motion (for DC Brushless motors only). To commute the current correctly the controller must have information about the electrical angle of the motor. This information can be received from Hall sensors, for example.

In the absence of Hall sensors, the electrical angle of the motor must be derived from the encoder readings, which are generally not aligned with the motor electrical angle. As a result, an initialization process is required in order to align the encoder reading and the motor's electrical phase.

This can be done in one of several methods.

ComtMode[] is an array that controls the electrical angle detection process.

ComtMode[1] selects which method will be used to detect the electrical angle:

- 0 – Jump to zero. The motor will jump from its present location to the electrical zero angle of one of the phases. This method usually involves a large, sudden movement.
- 1 – Minimal motion. In this method the motor will only perform a very small movement, and detect the electrical angle in its present location.
- 2 – Absolute encoder. For absolute encoder, perform angle detection once per each motor + controller set using one of the methods above. After the angle is detected use the Save command to save all the parameters. The absolute location of angle 0 will be saved in the controller's flash memory. In all the following uses of the same motor – controller set, use ComtMode[1]=2 to calculate the current electrical angle using the number saved in memory without any motion.

When using the "Jump to zero" method the user can determine the amount of voltage that will be applied to the motor during the process. If the voltage (and therefore the resulted current) is too low, mechanical factors in the system (such as friction) may prevent the motor from moving and cause failure of the process. High currents that are applied to the same phase for a long time can cause damage. The user should apply a current that is enough to move but not too much. The voltage (and thus the current) to the motor is increased gradually over a period of time so the maximum current is only applied for a short time. Every voltage (current) is applied for a step duration of 10mSec.

ComtMode[2] : Used only in "jump to zero" mode. Determines by how much the voltage to the motor is increased after each step. The resulting current depends on the bus voltage and the motor resistance. To prevent damage it is recommended to set this number low during the first experiments and raise it only if necessary. If needed change ComtMode[3].

ComtMode[3] determines how many voltage steps will be applied during the angle detection process. Each voltage value is applied to the motor for 10 mSec and then the voltage is increased. It is recommended to start with a 1 second detection process (ComtMode[3] = 100). After the detection is done, with motor disabled, move the motor manually to a different position and repeat the process. The motor is expected to move. If you don't see any motion, increase ComtMode[3] and move the motor again until you see a detection process that causes movement. As long as you don't see motion it is possible that the current is not enough to overcome friction or the motor was in the 0 position in the first place.

#### About the “Minimal motion” method:

With incremental encoder (or first using a motor with absolute encoder), there is no information about the electrical angle of the motor after power on. The “Jump to zero” method overcomes this difficulty by forcing the motor to jump into a known electrical angle (0 degrees). But this involves a relatively large motor’s jump. The “Minimal motion” method performs a different process, to minimize this motor movement.

The idea in general is to temporarily, during the process (about 2-3 seconds), to close a control loop that tries to maintain the motor at its current position, while commanding the current control loop with a fixed current command (set by the user) and an electrical angle that is the output of the control loop. As a result, the control loop (trying to keep the motor at its position when the process was started) will find the suitable electrical angle that will keep the motor in this location although current is applied to the motor.

This is, by definition, the electrical angle of the motor position, minus 90 degrees.  
So, the control loop finds the electrical angle of the motor, at its current position.

Since this angle is not known initially (a zero angle is assumed), the motor will slightly move, till the control loop will be stabilized into steady state, where the motor will be back in its original position.

In order to optimally overcome frictions and to ensure the accuracy of the process, after this phase (in closed loop) is completed, a second phase is performed. In this phase, a user defined current (higher than the one used in the closed loop) is applied in open loop to the motor, to force it into the position that is related to the detected angle.

Knowing the detected angle, and its related position, the drive can initiate the commutation variables (where is the “zero” of the electrical cycle) and is ready for motions.

When using the “Minimal motion” method the user shall determine the amount of current that will be used during the process, as follows:

ComtMode[6]: Is the current to use, in [mA], during the process of looking for the electrical angle in closed loop, as explained above. Typically, it is around 10% to 20% of the motor’s peak current. It must be above the current that is required to overcome the system friction.

ComtMode[7]: Is the current to use, in [mA], during the last phase of the minimal motion commutation process. In this phase, the current is applied in open loop, so that the motor is forced to the position that is related to the detected electrical angle. Typically, it is around 15% to 30% of the motor’s peak current. It must be above the current that is required to overcome the system friction and for optimal performance, it shall be higher than ComtMode[6].

Note that while ComtMode[6] and [7] are used to set the current reference (CurrRef) during the process, the actual value of CurrRef is subjected to the drive current limitations, as defined by the user.

As the “Minimal motion” method involves a closed loop control to look for the electrical angle at the current position, it involves gain and integral terms parameters for the closed loop, as explained above. These parameters shall be set by the user as follows:

ComtMode[8]: Is the gain of the PI control filter. Typical value is 5000. Changes may be required depending on the system, the encoder resolution and the motor’s number of poles.



ComtMode[9]: Is the integral gain of the PI control filter. Typical value is 20. Changes may be required depending on the system, the encoder resolution and the motor's number of poles.

ComtMode[4] holds the position of the electrical zero of a motor with absolute encoder. If the motor or controller need to be replaced, the electrical angle of the motor must be detected again and saved. The value of ComtMode[4] is automatically assigned following a commutation process with ComtMode[1]=0 or 1. This means that for a motor with absolute encoder, you shall first perform the detection process using ComtMode[1] = 0 (or 1) and then, after the process is successfully completed and ComtMode[4] is automatically updated by the controller, you can change to ComtMode[1]=2 ("Absolute encoder" mode) and save to the Flash, so that there will be no need for any motion in the next times.

ComtMode[5] is used to request a new commutation process. To repeat the commutation process enter ComtMode[5] = 1282. A new commutation process will begin and the value of ComtMode[5] will be cleared.

Commutation process is automatically performed following power on or reset.

Specific products notes:

1. Product 1:
  - a. Currently, only the "Jump to zero" method is supported.
  - b. Repeating the commutation process is done by ComtMode[1]=1282 (and not ComtMode[5]).

Description	Value
<b>Mnemonic (keyword):</b>	ComtMode
<b>CAN code:</b>	72
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 24
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ComtStatus](#),

## ComtStatus

Description	Value
<b>Mnemonic (keyword):</b>	ComtStatus
<b>CAN code:</b>	143
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ComtMode](#),

## ConFlt

ConFlt reports the error that caused the disabling of the motor. Every fault that is entered to ConFlt is also logged in ErrLog.

ConFlt is cleared by the next motor enable.

The table below shows the values of ConFlt and their meanings:

Value	Meaning
0	No fault has occurred
1001	Abort signal was detected
1002	Short from motor phase to ground
1003	The encoder is disconnected
1004	FPGA watchdog not received
1005	PWM dead time too short
1006	Hall input disconnected
1007	Motor is stuck
1008	Bus Voltage too high
1009	Bus Voltage too low
1010	Logic Voltage too high
1011	Logic Voltage too low
1012	Bus current too high
1013	Phase A current too high
1014	Phase B current too high
1015	Phase C current too high
1016	Motor current too high
1017	Driver power exceeds limit
1018	IPM temperature too high
1019	Velocity too high
1020	Position error exceeds limit
1021	Velocity error exceeds limit
1022	CPU temperature too high
1023	Bus Voltage too high - exceed absolute limitation
1024	STO1 activated
1025	Over current detected
1026	Auxiliary encoder disconnected
1027	IPM fault
1028	The selected encoder type is currently not supported
1029	The selected auxiliary encoder type is currently not supported

Description	Value
<b>Mnemonic (keyword):</b>	ConFlt
<b>CAN code:</b>	31
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ContCL

The ContCL (Continuous current limit, in [mA]) is used, together with PeakCL and PeakTime parameters, to define the behavior of the amplifier/motor I2t power limitation scheme. Please refer to the PeakCL keyword page for detailed description of the limitation scheme and the effect of ContCL.

Description	Value
<b>Mnemonic (keyword):</b>	ContCL
<b>CAN code:</b>	51
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[PeakCL](#), [PeakTime](#), [MotorCurr](#) and [StatReg](#).

## ControlMode

The ControlMode configuration parameter controls the operation of some special control algorithms.

Each bit at ControlMode turns on (or off) a given special algorithm, as follows:

Bit 0 (LSB):    "0"    -       The enhanced speed range algorithm is tuned off.  
                  "1"    -       The enhanced speed range algorithm is turned on.  
                  Using this algorithm, the controller pushes the supported speed range by +15%,  
                  without hitting the voltage (PWM) saturation.

Bit 1:           "0"    -       The current control scheme is based on vector control.  
                  "1"    -       The vector control is turned off, and the current control is closed  
                  on Ia, and Ib.  
                  With vector control (the default), much better current control (current command  
                  tracking) is achieved also at high speeds and high number of motor's pole pairs. In  
                  addition, there is much less non-effective current at the motor (especially at high  
                  speeds). Effective current is the current that generates torque (in contrast to  
                  current that just generates heat at the motor, but no torque).

Bit 2:           "0"    -       The control scheme includes current control.  
                  "1"    -       The control scheme does not include current control.  
                  The output of the velocity controller is used to drive the amplifier.  
                  This option is available only at some of Agito's controllers (especially the low  
                  power units). It is useful for low power motors, such as VCM, to provide better  
                  tracking and point to point performance.

Description	Value
<b>Mnemonic (keyword):</b>	ControlMode
<b>CAN code:</b>	109
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	7
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## CounterDown

Each member of CounterDown array is initialized to 0. The user can assign a starting value to this counter, and then the counter will count down each control sample until the value reaches 0. This is mainly useful for user programs.

Description	Value
<b>Mnemonic (keyword):</b>	CounterDown
<b>CAN code:</b>	39
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[CounterUp\[\]](#), [Time](#)

## CounterUp

Each member of CounterUp array is initialized to 0 (at power on or after reset) and will add 1 to its value every control sample. The user can assign a starting value to this counter, and then the counter will count up each control sample. This is mainly useful for user programs.

Upon reaching the maximal (32 bits) value of 2,147,483,647 the counter is rolling over to - 2,147,483,648 and continue counting.

Description	Value
<b>Mnemonic (keyword):</b>	CounterUp
<b>CAN code:</b>	40
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CounterDown\[\]](#), [Time](#)



## CurrAlnTh

A threshold value for going to Current Operation Mode if there is analog input defined as "Force Feedback" (see AlnMode[] above) and if its value is above the threshold.

If its value is 0, the change is not even checked for.

Positive and negative values are handled similarly to the definition in CurrPosErrTh.

Upon such change, its value is cleared, to avoid future undesired changes. User need to re-set its value for the next switch.

Description	Value
<b>Mnemonic (keyword):</b>	CurrAlnTh
<b>CAN code:</b>	338
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-100,000
<b>Max value:</b>	100,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ForceAlnTh](#), [GoToCurrMode](#),

## CurrCmdCntr

Counting the time, in msec within each entry of the array (when using User Defined values/time). User can set it to 0 (or any other value) before manually entering the current operation mode or enabling the motor. For automatic switching into Current Operation Mode (see later within this document), this parameter will be set to 0 upon the switching, by the controller.

Description	Value
<b>Mnemonic (keyword):</b>	CurrCmdCntr
<b>CAN code:</b>	334
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	122,070,306
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrCmdIndex](#), [CurrCmdSrc](#), [CurrCmdVal](#), [CurrCmdTime](#),

## CurrCmdIndex

When using User Defined values/time (CurrCmdSrc=1 or 2), this parameter reports the currently active index to CurrCmdTime. User can set it to 1 (or any other value) before manually entering the Current Operation Mode or enabling the motor. For automatic switching into Current Operation Mode (see later within this document), this parameter will be set to 1 upon the switching, by the controller.

Description	Value
<b>Mnemonic (keyword):</b>	CurrCmdIndex
<b>CAN code:</b>	333
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	20
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrCmdCntr](#), [CurrCmdSrc](#), [CurrCmdVal](#), [CurrCmdTime](#),

## CurrCmdSrc

This parameter is used to set the source if current command. Defined as following:

- 0: Analog (as defined in AlnMode).
- 1: User defined values/time.
- 2: User defined values/time, interpolated (future). For now, CurrCmdSrc=2 yields with the same as 1.

Description	Value
<b>Mnemonic (keyword):</b>	CurrCmdSrc
<b>CAN code:</b>	330
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrCmdCntr](#), [CurrCmdIndex](#), [CurrCmdVal](#), [CurrCmdTime](#),

## CurrCmdTime

Command time for each current command.

The unit for this command is msec.

Description	Value
<b>Mnemonic (keyword):</b>	CurrCmdTime
<b>CAN code:</b>	332
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-122,070,306
<b>Max value:</b>	122,070,306
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrCmdCntr](#), [CurrCmdIndex](#), [CurrCmdVal](#), [CurrCmdSrc](#),

## CurrCmdVal

Description	Value
<b>Mnemonic (keyword):</b>	CurrCmdVal
<b>CAN code:</b>	331
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrCmdCntr](#), [CurrCmdIndex](#), [CurrCmdVal](#), [CurrCmdTime](#), [GoToCurrMode](#),

## CurrDir

This functionality is currently not supported.

Description	Value
<b>Mnemonic (keyword):</b>	CurrDir
<b>CAN code:</b>	76
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

## CurrGain

Agito controllers implement PI control filter for the current closed loop control.

Two control loops are implemented. One for the current of motor phase A (Ia) and one for the current of motor phase B (Ib).

CurrGain is the gain of this PI.

The relevant equations are:

CurrRef = ... the output of the velocity control filter. Refer to the documentations of VelGain, VelKi.

IaRef = commutation (CurrRef)

IaErr = IaRef – Ia

IaIntegral = IaIntegral + IaErr \* CurrKi \* 0.001

IaIntegral is then saturated by the parameter MaxPWM

Temp = IaIntegral + IaErr

Va = Temp \* CurrGain \* 0.001

Va is then saturated by the parameter MaxPWM

(a similar control loop is executed for the current of motor's B phase – of course, with properly shifted commutation).

CurrRef, IaRef, Ia, IaErr are all in [mA]

Va is in internal units, with a product dependent value that represents 100% PWM duty cycle.

Please consult Agito if the exact internal value for the product you are using, is required.

Description	Value
<b>Mnemonic (keyword):</b>	CurrGain
<b>CAN code:</b>	104
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	200,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrKi](#), [Ia](#), [Ib](#), [CurrRef](#), [IaRef](#), [IbRef](#), [IaErr](#), [IbErr](#), [Va](#), [Vb](#), [Vc](#), [MaxPWM](#) and [ControlMode](#).



## CurrKi

Agito controllers implement PI control filter for the current closed loop control.

Two control loops are implemented. One for the current of motor phase A (Ia) and one for the current of motor phase B (Ib).

CurrKi is the gain of the integral term of this PI.

The relevant equations are:

CurrRef = ... the output of the velocity control filter. Refer to the documentations of VelGain, VelKi.

IaRef = commutation (CurrRef)

IaErr = IaRef – Ia

IaIntegral = IaIntegral + IaErr \* CurrKi \* 0.001

IaIntegral is then saturated by the parameter MaxPWM

Temp = IaIntegral + IaErr

Va = Temp \* CurrGain \* 0.001

Va is then saturated by the parameter MaxPWM

(a similar control loop is executed for the current of motor's B phase – of course, with properly shifted commutation).

CurrRef, IaRef, Ia, IaErr are all in [mA]

Va is in internal units, with a product dependent value that represents 100% PWM duty cycle.

Please address Agito if the exact internal value, for the product you are using, is required.

Description	Value
<b>Mnemonic (keyword):</b>	CurrKi
<b>CAN code:</b>	105
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	200,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrGain](#), [Ia](#), [Ib](#), [CurrRef](#), [IaRef](#), [IbRef](#), [IaErr](#), [IbErr](#), [Va](#), [Vb](#), [Vc](#), [MaxPWM](#) and [ControlMode](#).

## CurrLimFwd

Refer to the detailed description of the Current Limitation under the [CurrLimMode](#) keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	CurrLimFwd
<b>CAN code:</b>	393
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrLimMode](#), [CurrLimRev](#), [PeakCL](#),

## CurrLimMode

---

Current Limit (or Torque Limit) is the function of limiting the motor's current (actually, the limit is applied on the command to the current control loop).

In Agito controllers/drives the current command is always limited by the PeakCL (Peak Current Limit) parameter.

However, additional limitation mechanisms are supported:

1. General:

- a. The user can define additional mode of torque command (CurrRef) limitations.
- b. This additional mode is valid for all operational modes (Position, Velocity and Current Only).
- c. This mode includes limitations of the current (torque) command by analog inputs or by programmable parameters.
- d. The analog inputs (AInPort[1] and AInPort[2]) can be processed with : offset, gain, filter and dead band as explained under the AInPort page.

2. Relevant parameters:

1. Parameters: CurrLimMode, CurrLimFwd and CurrLimRev.
2. Behavior:

- a. If CurrLimMode == 0:  
No limits are applied (beside PeakCL).
- b. If CurrLimMode == 1:  
Analog inputs are used.  
AInPort[1] is used for forward limitation.  
AInPort[2] is used for reverse limitation.

Note that the analog inputs are used as absolute values (assuming that both are positive).

- c. If CurrLimMode == 2:  
Only Analog input 1 is used.  
AInPort[1] is used for forward limitation.  
AInPort[1] is used also for reverse limitation.
- d. If CurrLimMode == 3:  
Using the programmable parameters for limitations.

CurrLimFwd is used for forward limitation.  
CurrLimRev is used for reverse limitation.

Description	Value
<b>Mnemonic (keyword):</b>	CurrLimMode
<b>CAN code:</b>	392
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrLimFwd](#), [CurrLimRev](#), [CurrLimMode](#), [PeakCL](#),

## CurrLimRev

Refer to the detailed description of the Current Limitation under the [CurrLimMode](#) keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	CurrLimRev
<b>CAN code:</b>	394
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PeakCL](#), [CurrLimFwd](#), [CurrLimMode](#),

## CurrPosErrTh

A threshold value for going to Current Operation Mode if position error is greater than the threshold.

If its value is 0, the change is not even checked for.

If the value is positive, the trigger is made upon PosErr>value. If the value is negative, the trigger is made upon PosErr<value (to support positive or negative direction of motion).

Upon such change, its value is cleared, to avoid future undesired changes. User need to re-set its value for the next switch.

Description	Value
<b>Mnemonic (keyword):</b>	CurrPosErrTh
<b>CAN code:</b>	337
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-327,680
<b>Max value:</b>	327,680
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosErr](#), [GoToCurrMode](#),

## CurrRef

CurrRef returns the current reference generated by the internal control loops in milliAmperes.

Description	Value
<b>Mnemonic (keyword):</b>	CurrRef
<b>CAN code:</b>	26
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrCmdVal](#),

## DCDC

The DCDC[] array parameter reports some measured internal supply voltages, as follows:

DCDC[1]	-	3.3v
DCDC[2]	-	15v
DCDC[3]	-	-15v
DCDC[4]	-	1.2v
DCDC[5]	-	1.8v

The values are reported in [mV].

Description	Value
<b>Mnemonic (keyword):</b>	DCDC
<b>CAN code:</b>	42
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 6
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## DebugData

This parameter is for development team use only. Using this parameter may result in unexpected behavior.

Description	Value
<b>Mnemonic (keyword):</b>	DebugData
<b>CAN code:</b>	224
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 199
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Decel

Decel is the deceleration used for all motions in user units / sec<sup>2</sup>. This deceleration is also used with pulse – direction command input when the indirect mode is applied.

Decel can be changed during motion and the latest value will be effective immediately. If the current motion is still decelerating, the deceleration rate will be changed. Otherwise the new acceleration value will be used the next time acceleration is required.

Decel is also used for deceleration after a Stop command was entered.

For emergency stops EmrgDec is used.

Description	Value
<b>Mnemonic (keyword):</b>	Decel
<b>CAN code:</b>	137
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EmrgDec](#), [Stop](#)

## DInFilt

The value of DInFilt determines how the pulses that are entered through the pulse and direction input will be filtered.

The pulse direction input is sampled at 80MHz. If DInFilt number of consecutive readings are identical this value is latched. For example, if DInFilt = 4, 4 consecutive readings of logic "1" are needed before the read value is changed from 0 to 1.

Description	Value
<b>Mnemonic (keyword):</b>	DInFilt
<b>CAN code:</b>	213
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	15
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## DInLog

DInLog effects the login of the digital input reading. If DInLog = 0, DInPort will return "1" for each input in "On" state, and "0" for each input in "Off" state. If the corresponding bit in DInLog is "1". DInPort logic is inverted: "0" returned for "On" and "1" returned for "Off".

Example:

If DInLog = 4 (Invert the logic for input 3), and the input state is as follows:

Input 1 = On

Input 2 = Off

Input 3 = Off

All the rest = Off

DInPort should return "1" in bit 0, "0" in bit 1, and "1" in bit 2 (input 3 is off, but it is inverted).

DInPort = 5

Description	Value
<b>Mnemonic (keyword):</b>	DInLog
<b>CAN code:</b>	214
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[DInPort](#), [DInFilt](#), [DInMode](#)

## DInMode

DInMode[] is an array the size of the digital inputs port. The value of DInMode[1] determines the special use (if any) of input 1 (DInPort bit 0, which is the right-most bit), the value of DInMode[2] determines the special use of input 2 (DInPort bit 1) and so on.

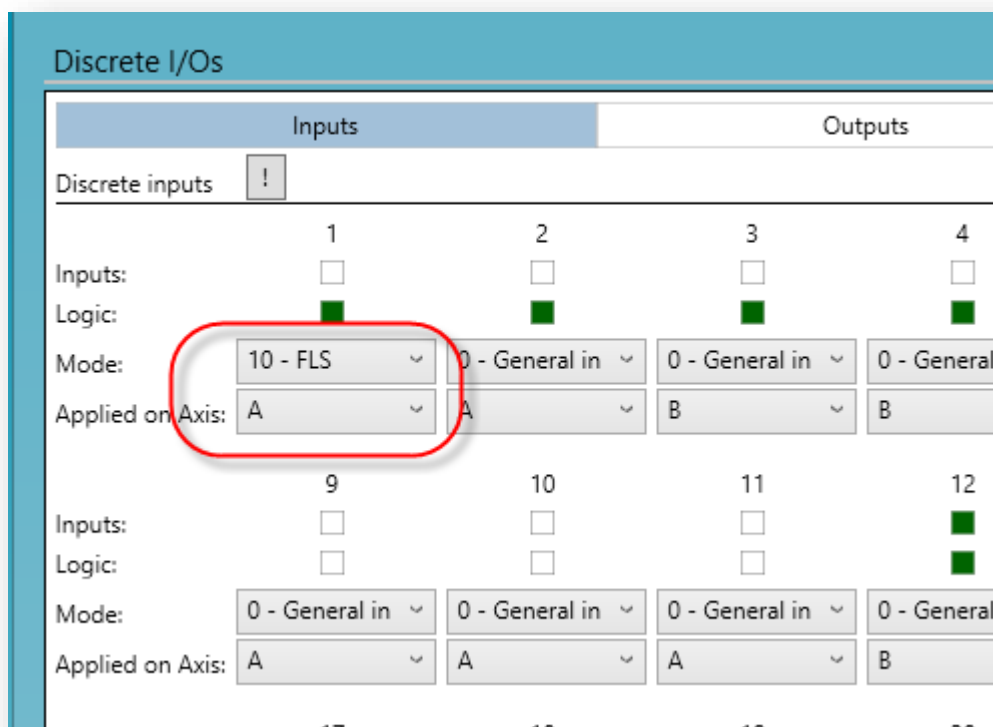
The value of the input port bits can be always read at DInPort, independently of the use of each specific bit by a special function, as set by DInMode[]. The right most bit (bit 0) reflects the value of input 1, the next bit is input 2 and so on.

For easy use it is recommended to set the input modes using the PC suite's drop down menu.

Note that the value of DInMode[i] is composed by the special use (if any) that is given to input "i" and the axis on which this special use is applied to, as follows:

Lower 16 bits are used to define the special use, as listed below, and the upper 16 bits (bits 16 to 31) are used to define the axis (or axes) on which this special use is applied. If bit 16 is "1", the special use will apply to axis A, bit 17 is linked to axis B and so on. Multiple axes can be selected.

For example:



Here input 1 is assigned with a special use (function) that is FLS (Forward Limit Switch) of Axis A.

The value of DInMode[1] will be: 65546, which is: 10 (means: FLS) + (0x0001 << 16) which means bit 16 is "1" to indicate A Axis.

Notes:

1. It is highly recommended to reset the controller after making changes to DInMode[] since some of the special functionalities require reset in order to work (or stop working) properly. Remember to save your changes to the Flash before resetting the controller.
2. It is not allowed to set more than 20 special functions to the discrete inputs. In case more than 20 functions are set, only the first 20 will be operational. Note that if a special function is assigned to a given input and is set to be applied on two axes, for example, it is counted as 2 special functions.

The possible values for each entry of DInMode[] and the corresponding functions are listed in the table below:

Value

Input Mode.

0

General input:

General purpose input, no special function is assigned to this input, frequently used in user programs.

1

Dedicated HW function:

Not in use.

2

Motor on:

The motor is enabled if this input is set high while the motor is off and alarm status is cleared (see item 7 below). The motor is immediately disabled if this input is cleared.

If no input is assigned with this functionality, the motor will be enabled or disabled according to the MotorOn message over the communication and the status of Alarms.

This input is for PD motion mode and is doing motor on + begin.

3

Begin motion:

Begin motion upon rising edge of this input.

This function works together with the BeginDInOn parameter. If BeginDInOn is "1", and after Begin was commanded, a rising edge at this input will start the motion.

If no input is assigned with this functionality, motion cannot be started using a discrete input.

4

Stop motion:

Stop motion upon rising edge of this input. Equivalent to the Stop message over the communication.

If no input is assigned with this functionality, motion cannot be stopped using a discrete input.

5

Clear input pulses:

Suitable for Pulse/Direction motion mode. When the motor is on and this input is set (rising edge) stop the motion and disable the motor. On a falling edge begin a new motion (will again move according to the incoming pulse/direction).

If no input is assigned with this functionality, this special function is not accessible.

6

Abort and resume motion:

Not in use. See item 5 above.

7

Reset/Clear Fault/Alarm:

To function, the fault/alarm reset/clear signal must be on for at least 20 msec. Once cleared (value of "0") after being enabled ("1") for at least 20ms, the alarm state is cleared. The ConFlt status parameter is cleared. If an output was assigned the functionality of Alarm, this output is reset.

If no input is assigned with this functionality, the Alarm status can be cleared over the communication by re-enabling the motor: MotorOn=1.

8

Abort motion:

Abort the motion when this input is set. Equivalent to the Abort message over the communication.

If no input is assigned with this functionality, motion cannot be aborted using a discrete input.

9

RLS:

Reverse limit switch. If the motor is moving in the negative direction it will stop when this input goes high.

If no input is assigned with this functionality there will be no reverse hardware limit switch in the system.

10

FLS:

Forward limit switch: If the motor is moving in the positive direction it will stop when this input goes high.

If no input is assigned with this functionality there will be no forward hardware limit switch in the system.

11

Torque limit on:

The torque (current) limits are controlled using the parameter CurrLimMode. This control is disabled (equivalent to CurrLimMode = 0) when this input is cleared and function normally (according to the value of CurrLimMode) is this input is set (or if no input is assigned with this functionality).

12

Activate dynamic break:

This input is directly linked to the parameter DynBrakeOn. If the input is set, the parameter is set to 1. If the input is cleared, the parameter is cleared (set to 0). If no input is assigned with this functionality, the parameter will get the value set for it over the communication or as saved in the Flash memory.

The dynamic brake is activated based on the value of DynBrakeOn and some other conditions as described under the DynBrakeOn documentation.

13

Activate motor brake:

When this input is set, the discrete output that is defined (if any) for motor brake functionality is set. When the input is cleared, the output is cleared as well.

If no input is assigned with this functionality, the relevant output can be set or cleared over the communication using the DOutPort parameter.

14

Change control filter set:

Generally, the currently active control set (PosGain, VelGain, VelKi and AccFFW) are controlled using the ScheduleMode parameter (see documentation). However, when ScheduleMode is equal to 1 (Manual/DInPort mode), the currently active control set can be controlled using this input.

If the input is cleared (0), the active control set is set to set 1. If the input is set (1), the control set is set to 2. The currently active set can be monitored at the parameter ScheduleSet.

If no input is assigned with this functionality and the ScheduleMode is equal to 1 (Manual/DInPort mode), the active control set can be assigned using the ScheduleSet parameter over the communication.

15

Add velocity filter:

When this input is set (1), the bi-quad velocity filter 2 is executed normally. When this input is cleared, the filter is not executed and is bypassed. It is recommended to switch the filter on again during motor off condition.

If no input is assigned with this functionality, the velocity filter 2 is executed normally.

Note:

1. We can achieve on-the-fly enable or disable velocity filter 2 by changing the Dinport value;
2. AVelFiltOn[2] = 1 can not tell you the filter is always executed when you set the DinMode == 'Add Velocity Filter'

16

Velocity/Position mode switch:

The operational mode (OperationMode parameter) is directly controlled by this input.

If the input is cleared (0), the OperationMode is set to 2 (Velocity control). If the input is set (1), the OperationMode is set to 3 (Position Control).

The mode change happens only when the motor is off. The input has no effect when the motor is enabled.



If no input is assigned to this functionality (and to the following items 17 and 18), OperationMode will have the value as set over the communication or as read from the Flash memory.

17

Velocity/Current mode switch:

The operational mode (OperationMode parameter) is directly controlled by this input.

If the input is cleared (0), the OperationMode is set to 1 (Current control). If the input is set (1), the OperationMode is set to 2 (Velocity Control).

The mode change happens only when the motor is off. The input has no effect when the motor is enabled.

If no input is assigned to this functionality (and to the item 16 above and 18 below), OperationMode will have the value as set over the communication or as read from the Flash memory.

18

Position/Current mode switch:

The operational mode (OperationMode parameter) is directly controlled by this input.

If the input is cleared (0), the OperationMode is set to 1 (Current control). If the input is set (1), the OperationMode is set to 3 (Position Control).

The mode change happens only when the motor is off. The input has no effect when the motor is enabled.

If no input is assigned to this functionality (and to items 16 and 17 above), OperationMode will have the value as set over the communication or as read from the Flash memory.

19

Clear absolute encoder:

Not in use.

20

Change speed:

This input will be used for on-the-fly change of the Speed keyword. Upon rising edge of this input (change from "0" to "1"), the value of the SpeedChgNew parameter of the assigned axis, will be pushed into the Speed parameter of this axis.

21

Home:

Indicates that this input is the Home switch of the assigned axis. The Home switch is used by some of the supported Homing sequences (refer to HomingOn keyword documentation).

22

Mode switch between Position Operation Mode and Force Operation Mode:

Rising edge ("0" to "1") of this input will perform on-the-fly operation mode switch (for the assigned axis), from Force Operation Mode to Position Operation Mode. If the assigned axis is not in Force Operation Mode, the rising edge will have no effect.

Falling edge ("1" to "0") of this input will perform on-the-fly operation mode switch (for the assigned axis), from Position Operation Mode to Force Operation Mode. If the assigned axis is not in Position Operation Mode, the falling edge will have no effect. On-the-fly switch to Force Operation Mode is done identically to the GoToForceMode keyword.

The falling edge will have no effect if the assigned axis is current participating in a CNC motion.

23

Hall A:

Indicates that Hall A (the first hall signal) of the assigned axis is connected to this input. The controller assumes that Halls B and C (second and third signals) are connected to the next inputs (needs to be defined as General Inputs without any special functionality).

Note that the hall effects can be connected at any order and the controller supports a learning process (if requested by the user) to learn and adjust itself to the order of the hall effect connections.

24 (supported only from FW version 1.3.0)

Fault input:

The input is used as an indication of external fault situation. If the input is "1" and if the motor of the assigned axis is enabled (and not in simulation mode), the motor will be disabled with a relevant error code. This function is mainly used when working with external drive (for example over a  $\pm 10V$  command).

A value of "0" at this input has no effect (no fault).

Note:

If few inputs are assigned with the same special functionality, the higher input (e.g.: Input 5 is higher than input 3) will be internally used for this functionality.

Description	Value
<b>Mnemonic (keyword):</b>	DInMode
<b>CAN code:</b>	225
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 22
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	4,128,791
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[DOutMode\[\]](#), [DInPort](#), [DInLog](#), [DOutPort](#)

## DInPort

DInPort returns a decimal value that represents the status of the digital input port. Translate the value returned by DInPort to binary to see the state of each individual input.

Bit 0 (LSB) of DInPort corresponds to input 1.

Bit 1 of DInPort corresponds to input 2 and so on, according to the actual number of digital inputs supported.

Note that DInLog controls the logic of the inputs. If DInLog = 0, DInPort will return "1" for each input in "On" state, and "0" for each input in "Off" state. If the corresponding bit in DInLog is "1", DInPort logic is inverted: "0" returned for "On" and "1" returned for "Off".

Example:

If DInLog = 4 (Invert the logic for input 3), and the input state is as follows:

Input 1 = On

Input 2 = Off

Input 3 = Off

All the rest = Off

DInPort should return "1" in bit 0, "0" in bit 1, and "1" in bit 2 (input 3 is off, but it is inverted).

DInPort = 5

Description	Value
<b>Mnemonic (keyword):</b>	DInPort
<b>CAN code:</b>	34
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[DInLog](#), [DInFilt](#), [DInMode](#)

## DoNothing

This command is for Agito internal use only. If used, as implied by its name, it will do nothing.

Description	Value
<b>Mnemonic (keyword):</b>	DoNothing
<b>CAN code:</b>	239
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## DOutLog

DOutLog can be used to invert the logic of the digital outputs. If the bit in DOutLog that corresponds to an output is 0, the logic of that input is unchanged. 1 in DOutPort will result in an output in “on” condition, and 0 will be “off”.

If the corresponding bit in DOutLog is 1 the logic of the output is inverted: 1 will result in “off” and 0 will be “on”.

Note that bit 0 (LSB) of DOutLog corresponds to input 1.

Description	Value
<b>Mnemonic (keyword):</b>	DOutLog
<b>CAN code:</b>	212
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[DoutPort](#), [DOutMode](#), [DOutType](#)

## DOutMode

DOutMode is an array the size of the digital outputs port. The value of DOutMode[1] determines the special use (if any) of output 1 (DOutPort bit 0, which is the right-most bit), the value of DOutMode[2] determines the special use of output 2 (DOutPort bit 1) and so on.

If a given output bit is assigned with a special function (the related DOutMode[] entry is not zero), the related bit at DOutPort will be set or cleared by this functionality and writing to this bit over the communication (using DOutPort) will have no effect (or actually a very short term effect, till the special function will be re-calculated, typically each control interrupt).

If a given output bit is not assigned with a special function (the related DOutMode[] entry is 0), then the output is controlled by the related bit at DOutPort.

For easy use it is recommended to set the output modes using the PC suite's drop down menu. The possible values for each entry of DOutMode[] and the corresponding functions are listed in the table below:

Value

Output mode functionality

0

General output:

This input has no special functionality. It is controlled by the relevant bit at DOutMode.

1

Dedicated HW function:

Not in use.

2

Motor on:

The output reflects the status of the motor (actually, the value of the MotorOn status parameter).

3

In Motion:

The output is set if the motor is in motion and is cleared otherwise. Actually, the output reflects the value of MotionStat status parameter (this parameter is zero when not in motion and non-zero when in motion).

4

In Acceleration:

The output is set if the motor is accelerating and is cleared otherwise (reflecting the suitable bit at MotionStat).

This functionality is valid only for motion modes that use the built-in profiler (for example: indirect Pulse/Direction uses the profiler, while direct Pulse/Direction does not).

5

In Deceleration:

The output is set if the motor is decelerating and is cleared otherwise (reflecting the suitable bit at MotionStat).

This functionality is valid only for motion modes that use the built-in profiler (for example: indirect Pulse/Direction uses the profiler, while direct Pulse/Direction does not).

6

In constant speed:

The output is set if the motor is in constant speed (in motion, but not accelerating or decelerating) and is cleared otherwise.

This functionality is valid only for motion modes that use the built-in profiler (for example: indirect Pulse/Direction uses the profiler, while direct Pulse/Direction does not).

7

End of motion:

Not in use. See item 3 above and 8 below.

8

In target:

The output is set when the motor is “in-target” and is cleared otherwise. Actually, the output reflects the value of InTargetStat status parameter. The output is set when this parameter is equal to 4 (target reached) and is cleared otherwise.

9

Fault/Alarm status:

The output will be set when the unit is in Alarm state and will be cleared otherwise. This actually means that the output reflects the value of the ConFlt status parameter (0 if no Alarm, non-zero is Alarm).

10

Warnings found in last motion:

Currently not in use.

11

Saturations in last motion:

Currently not in use.

12

RLS or FLS are active:

The output is set if the RLS or the FLS inputs are activated and is cleared otherwise. This output actually reflects the value of the LimitsStat (zero or non-zero) status parameter.

13

Out of travel (RevPLim, FwdPLim):

The output is set if the position reference is higher than FwdPLim or smaller than RevPLim, and is cleared otherwise.

14

Regeneration is activated:

The output is set if regeneration is activated and is cleared otherwise.



15

Dynamic brake is activated:

The output is set if the dynamic brake is activated and is cleared otherwise.

16

Motor brake is activated:

The output is set if the motor brake is activated. This output can be tied to the motor brake.

No special function is executed here. This definition is used together with DInMode[i]=13 (see DInMode documentation). If no input is assigned with DInMode[i]=13, than this output is controlled using DoutPort.

Example:

To see the "MotorOn" status on output 4:

DOutMode[4] = 3

If the logic of output 4 is not inverted, output 4 will be on when the motor is enabled and off when it is disabled. You can use DOutLog to invert this logic.

Note:

If few outputs are assigned with the same special functionality, the higher output (e.g.: output 5 is higher than output 3) will be internally used for this functionality.

Description	Value
<b>Mnemonic (keyword):</b>	DOutMode
<b>CAN code:</b>	210
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 16
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	327,696
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[DInMode\[\]](#), [DoutPort](#), [DoutLog](#), [DInPort](#)

## DOutPort

The values of the bits in DOutPort affect the discrete output port of the controller as follows:

Bit 0 of DOutPort corresponds to output 1, bit 1 of DOutPort corresponds to output 2 etc.

The actual state of the output also depends on other related parameters:

The value of bit 0 of DOutPort is XORed with bit 0 of DOutLog.

The value of bit 1 of DOutPort is XORed with bit 1 of DOutLog.

And so on...

If DOutMode[x] is not 0, the output state of the relevant output will not be effected by the value of the relevant bit in DOutMode. The output will reflect only the special functionality.

### Notes:

In new FW versions (from FW version 1.0.2), there is an additional keyword that affects the state of the discrete output pin. This is DOutSelect. It does not affect the value of DOutPort, but it selects the source of the discrete output pin. A bit at DOutPort defines the state of the related discrete output only if the related DOutSelect[output number] is 0. See the DOutSelect documentation.

At old FW versions, the value of DOutPort was saved to the FLASH. Newer FW versions (from FW version 1.3.0) do not save DOutPort to the FLASH and instead initialize it to its default value (0). Still, user can affect the state of a discrete output pin at power up, by using the DOutLog parameter (which is saved to Flash).

Description	Value
<b>Mnemonic (keyword):</b>	DOutPort
<b>CAN code:</b>	211
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If DOutPort = 0x03

And DOutLog = 0x01

And all the values in DOutMode are 0 then:

The value of output 1 is inverted, since bit 0 of DOutLog is 1. Bit 0 of DOutPort is 1. The XOR of the bits is 0 (or you can think of it as 1 inverted) so output 1 will be off.

The value of output 2 is not inverted, since bit 1 of DOutLog is 0. Bit 1 of DOutPort is 1 and output 2 will be on.

If we now change DOutMode[1] to reflect the motor on state, the value of output 1 will be on when the motor is disabled and off when the motor is enabled, since the logic of this output is inverted. The value of the bit in DOutPort has no effect on the output state.

**See Also:**

[DOutMode](#), [DOutLog](#), [DOutType](#), [DOutSelect](#).

## DOutPortCBit

Used to clear a particular output.

Description	Value
<b>Mnemonic (keyword):</b>	DOutPortCBit
<b>CAN code:</b>	491
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 16
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

ADOutPortCBit[1]

**See Also:**

[DOutPortSBit](#), [DOutPortTBit](#), [DOutPort](#).

## DOutPortSBit

Used to set a particular output.

Description	Value
<b>Mnemonic (keyword):</b>	DOutPortSBit
<b>CAN code:</b>	490
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 16
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

ADOutPortSBit[1],

**See Also:**

[DOutPortCBit](#), [DOutPortTBit](#), [DOutPort](#).

## DOutPortTBit

Used to toggle a particular output.

Description	Value
<b>Mnemonic (keyword):</b>	DOutPortTBit
<b>CAN code:</b>	492
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 16
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

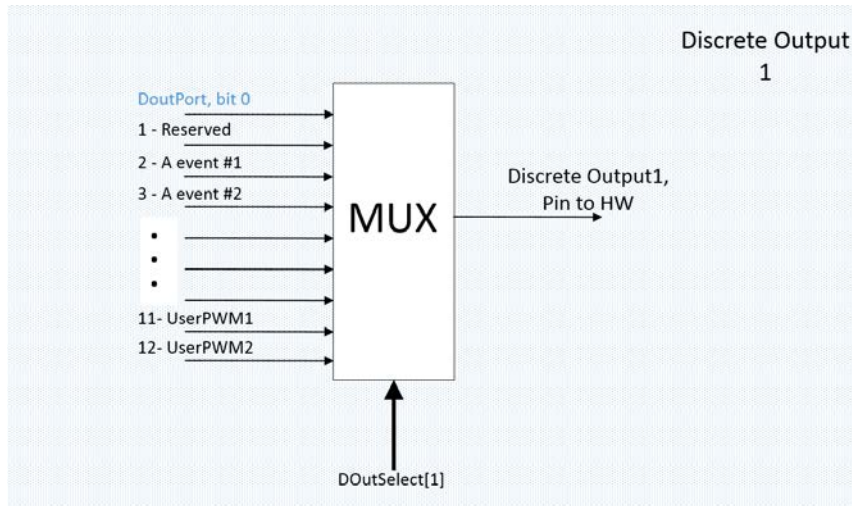
ADOutPortTBit[1],

**See Also:**

[DOutPortCBit](#), [DOutPortSBit](#),

## DOutSelect

There is a MUX before each discrete output pin. This MUX makes the discrete output pin can send out normal functionality output, and also can send out select output based on the setting of DOutSelect.



For example, if we set DOutSelect[1]=0. Then Digital OutPut 1 will be used as normal functionality output (controlled by the keywords: DOutPort, DOutLog, DOutMode, DOutPortSB and DOutPortCB). And if we set DOutSelect[1]=3, then it will be used as the Digital Output of “A event #2”.

DOutMode defines special functionality that is given to a given output bit and is implemented by the controller software. For example, a given output can represent the state of Motor On of a given axis. Setting/clearing this output bit is done automatically by the controller software, and the relevant bot at DOutPort is updated accordingly.

However, other types of special functions are signals that are created by the hardware and are generally too fast for software handling. Selecting these types of output functionalities is done using DOutSelect, which, using a MUX connects the selected hardware signal to the relevant output pin.

For AG300, the select output functions are listed as following:

0 – Software DOutPort

1 – Reserved or Pre-Defined function at differential outputs as following:

Differential output 1(DOutPort bit 4) – Encoder Emulation A for Axis A

Differential output 2(DOutPort bit 5) – Encoder Emulation B for Axis A

Differential output 3(DOutPort bit 6) – Encoder Emulation Z for Axis A

Differential output 4(DOutPort bit 7) – Encoder Emulation Pulse for Axis A

2 – A event #1

3 – A event #2

4 – A event #3

5 – B event #1

6 – B event #2

7 – B event #3

- 8 – C event #1
- 9 – C event #2
- 10 – C event #3
- 11 – UserPWM 1
- 12 – UserPWM 2

**Note:**

This list is specific for AG300, and only available from v1.3.0 and above of the firmware with FPGA version v1.3.0 and above.

For [Centrali](#), the select output functions are listed as following:

- 0 - Software DOutPort
- 1 - Reserved
- 2 - Main event #1
- 3 - Main event #2
- 4 - Main event #3
- 5 - Aux. event #1 (future)
- 6 - Aux. event #2 (future)
- 7 - Aux. event #3 (future)
- 8 - Pulse (P/D) (future)
- 9 – Direction (P/D) (future)
- 10 - Reserved
- 11 - Reserved
- 12 – Reserved

**Note:**

This list is specific for Centrali.

Description	Value
<b>Mnemonic (keyword):</b>	DOutSelect
<b>CAN code:</b>	314
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	12
<b>Default value:</b>	0
<b>User units:</b>	None
<b>Implementation status:</b>	Implemented

**Example:**

ADoutSelect[1]=0;

**See Also:**

[DOutPort](#), [DOutMode](#), [EventType](#), [UserPWM](#).



## DOutType

Some of the digital outputs of the controller can be configured as either sink or source. See the hardware user guide to find out which outputs and connection details.

To use an output as sink set the corresponding bit in DOutType to 0 (default). To use an output as source set the corresponding bit in DOutType to 1.

Note that bit 0 (LSB) corresponds to output 1.

Description	Value
<b>Mnemonic (keyword):</b>	DOutType
<b>CAN code:</b>	209
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	65,535
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[DOutMode](#), [DOutLog](#), [DOutPort](#)

## DownloadFW

DownloadFW will cause the controller to go into download firmware mode. Please use this option through Agito PC suite only. The password required is 160412.

For a user who wishes to program their own PC interface please contact Agito.

Description	Value
<b>Mnemonic (keyword):</b>	DownloadFW
<b>CAN code:</b>	230
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

## DownloadUPBin

Description	Value
<b>Mnemonic (keyword):</b>	DownloadUPBin
<b>CAN code:</b>	207
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## DPosRef

Description	Value
<b>Mnemonic (keyword):</b>	DPosRef
<b>CAN code:</b>	155
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[PosRef](#), [Pos](#),

## DualLoopFact

Please refer to the [DualLoopOn](#) keyword page for detailed description of the dual loop operation.

Description	Value
<b>Mnemonic (keyword):</b>	DualLoopFact
<b>CAN code:</b>	270
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	6,553,600
<b>Default value:</b>	65,536
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[DualLoopOn](#), [AuxPos](#) and [Vel](#).

## DualLoopOn

DualLoopOn is used to enable dual loop control structure.

If DualLoopOn == 0 (the default value), the dual loop control structure is disabled and the default control loop structure is used. In this mode, only the main encoder is used for both the position feedback (Pos) and for the velocity feedback (Vel).

If DualLoopOn == 1, the dual loop control structure is enabled. In this mode, the main encoder reading (Pos) is used for the position feedback and the auxiliary encoder reading (AuxPos) is used to derive the velocity feedback (Vel).

Note that although the auxiliary encoder speed is reported at AuxVel and the main encoder velocity is reported normally at Vel[1], Vel[2] and Vel[3], when the dual loop structure is used, [Vel](#) [1] (used as the feedback for the velocity control loop) is the filtered velocity of the auxiliary encoder and not of the main encoder.

This means that while typically Vel[2] is used as the input to the 1<sup>st</sup> velocity b-quadrant filter (whose output is Vel[1]), when dual loop is enabled, the AuxVel is used as an input to this filter.

Due to the built-in structure of the closed loop control filter (specifically, due to the built-in velocity feed-forward) the main encoder (used for position feedback) and the auxiliary encoder (used for the velocity feedback) must be properly scaled when dual loop is enabled.

Properly scaling means that if the motor is moving at a given velocity, the reading of Vel[2] and AuxVel will be identical. This is a function of the main and the encoder resolutions and the machine structure.

Scaling is supported using the [DualLoopFact](#) parameter.

The AuxVel is multiplied by DualLoopFact/65536 before it is used as the velocity feedback.

Description	Value
<b>Mnemonic (keyword):</b>	DualLoopOn
<b>CAN code:</b>	269
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[DualLoopFact](#), [AuxPos](#), [AuxVel](#) and [Vel](#).

## DualStuckTime

Description	Value
<b>Mnemonic (keyword):</b>	DualStuckTime
<b>CAN code:</b>	158
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	131,071,993
<b>Default value:</b>	250
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## DualStuckVel

Description	Value
<b>Mnemonic (keyword):</b>	DualStuckVel
<b>CAN code:</b>	157
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	40,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## DynBrakeOn

Dynamic braking function shorts the phases of the motor (all the three or two of them: drive dependent) when the drive is disabled and the motor velocity is below a given threshold. The dynamic brake function is activated when **all** the following conditions are satisfied:

- The parameter DynBrakeOn == 1.
- The drive is disabled.
- The motor velocity (absolute value) is below the DynBrakeSpeed parameter value.

Once the dynamic brake is activated, it will be deactivated upon **one of** the following conditions:

- The parameter DynBrakeOn == 0.
- The drive is enabled.

Note that the dynamic brake is not deactivated even if the motor speed becomes higher than the [DynBrakeSpeed](#) parameter value.

The dynamic brake status is reported at one of the bits of the [StatReg](#) parameter.

Description	Value
<b>Mnemonic (keyword):</b>	DynBrakeOn
<b>CAN code:</b>	405
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[DynBrakeSpeed](#), [StatReg](#),

## DynBrakeSpeed

Refer to the detailed explanation regarding the dynamic braking under the [DynBrakeOn](#) page.

Description	Value
<b>Mnemonic (keyword):</b>	DynBrakeSpeed
<b>CAN code:</b>	404
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	5,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[DynBrakeOn](#),

## ECAMCycCount

ECAMCycCount is a read only array that reports the number of the current cycle for each ECAM table.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMCycCount
<b>CAN code:</b>	307
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMCycles

ECAMCycles[x] determines how many times the cyclical part of the motion will be repeated for ECAM table x. If ECAMCycle[3] = 10 the part of the table between ECAMStartCyc[3] and ECAMEndCyc[3] will be repeated 10 times.

ECAMCycles can be set to a negative value to indicate that the motion is symmetrical around the starting point of the cycles, and the motion will continue the same number of repetitions in the negative and in the positive directions.

The parts of the table that are out of the boundaries of the cyclical part are normally used for acceleration or deceleration. If a negative value of ECAMCycles is used, the part of the motion that should happen when the most negative location is reached will be moved to its new logical location.

### Endless ECAM:

If ECAMCycles[x] is set to 2147483647 (exactly), the controller will perform infinite number of ECAM cycles, until some external event will stop the motion (such as: Stop or StopECAM commands are received, or hitting limit switches ...) or disable the motor (MotorOn=0 message or some fault).

For negative values of ECAMCycles[x], use a value of -2147483648 (exactly) to create endless ECAM.

Note that endless ECAM is possible only if selecting a master that can "work" forever. Useful examples are AuxPos and CounterUp[]. Both of them roll over at the maximal value of 32 bits. The endless ECAM is designed to provide smooth and transparent operation during this rolling over.

### Note:

When starting an ECAM motion, the software performs some calculations to protect from motion outside of allowed position range (32 bits). In order to pass these tests (and not to receive error message upon Begin motion), the master value needs to be with the range of -2,000,000,000 to 2,000,000,000 (actually even not close to these values). It is therefore recommended to set the master values (may it be AuxPos or CounterUp[]) to a value well within this range before starting an endless ECAM. Recommended value is 0.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMCycles
<b>CAN code:</b>	305
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#),  
[ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMEnd

ECAMEnd[] is the index of GenData[] array that holds the last element of the relevant table.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMEnd
<b>CAN code:</b>	303
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If table 1 is in locations GenData[1] to GenData[10], and table 2 is in locations GenData[11] to GenData[30] then:

ECAMEnd[1] = 10

ECAMEnd[2] = 30

### See Also:

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMEndCyc

ECAMEndCyc[x] is the index in Gendata[] array that holds the last location of the part of ECAM table x that should be cyclically repeated.

For example, if the values ECAM table 3 are in location 100 to 150 of the GenData array, and the part of the table between locations 120 and 140 should be repeated cyclically, then

ECAMEndCyc[3] = 140.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMEndCyc
<b>CAN code:</b>	302
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMGap

ECAMGap[x] determines the gap between master points for ECAM table x. The ECAM table that is stored in GenData holds only the values for the slave motor (what would be the “y” values of the function). It is assumed that the points along what we would think of as “x” axis are evenly spaced. ECAMGap defines the spacing between the points.

Note that negative values of ECAMGap can be used to invert the direction of the master reading.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMGap
<b>CAN code:</b>	304
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-8,000,000
<b>Max value:</b>	8,000,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)



## ECAMMaster

ECAMMaster[x] is the parameter to be used as the master value for ECAM table x. ECAMMaster is in complex CAN code. Any parameter that can be used in communication, including arrays, can be used as a master value for the ECAM motion.

The easiest way to determine ECAMMaster is by using the PC Suite that will calculate the value of the CAN code automatically.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMMaster
<b>CAN code:</b>	309
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMMasterIni

ECAMMasterIni[] is subtracted from the beginning position if needed to compensate for master motion.

Example:

If ECAMMasterIni[1] = 0 and the Begin command is received when the master position is 3000 then all master positions are relative to 3000. The ECAM motion will start immediately, giving position command to the position pointed by ECAMStart[1]

If ECAMMasterIni[1] = 2000 and the Begin command is received when the master position is 3000 then all master positions are relative to 1000. The motion will start immediately as if the master already moved to value 1000. The user must design the table so that this motion is reasonable.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMMasterIni
<b>CAN code:</b>	306
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMStart

ECAMStart[] is the index of GenData[] array that holds the first element of the relevant table.

Example:

If table 1 is in locations GenData[1] to GenData[10], and table 2 is in locations GenData[11] to GenData[30] then:

ECAMStart[1] = 1

ECAMStart[2]=11

Description	Value
<b>Mnemonic (keyword):</b>	ECAMStart
<b>CAN code:</b>	300
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMStartCyc

ECAMStartCyc[x] is the index in Gendata[] array that holds the first location of the part of ECAM table x that should be cyclically repeated.

For example, if the values ECAM table 3 are in location 100 to 150 of the GenData array, and the part of the table between locations 120 and 140 should be repeated cyclically, then ECAMStartCyc[3] = 120.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMStartCyc
<b>CAN code:</b>	301
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## ECAMTableNum

The controller can hold multiple ECAM tables. One of these tables is the active table at any time. ECAMTableNum selects which table is used for the current motion.

Example:

It is possible to have one table, referred to as Table 1 in locations 1-10 of GenData[] array, another table (Table 2) in locations 11-30 GenData[], and a third one in locations 31-55. Each table can have different properties: different gap, different number of cycles etc. All of these parameters can be saved in advance.

To use table 1 enter ECAMTableNum = 1.

When table 2 is needed, simply change to ECAMTableNum = 2.

Description	Value
<b>Mnemonic (keyword):</b>	ECAMTableNum
<b>CAN code:</b>	311
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	10
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMMaster](#), [StopECAM](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMCycCount](#), [ECAMMasterIni](#)

## EmrgDec

EmrgDec is the emergency deceleration in UsrUnits/Sec2. This parameter should normally be set to a much higher deceleration than decal. The value in EmrgDec is used when the motion has to be stopped by one of the following reasons:

Hardware limit

Software limit

Description	Value
<b>Mnemonic (keyword):</b>	EmrgDec
<b>CAN code:</b>	140
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[RevPlim](#), [FwdPlim](#), [Decel](#), [DInMode](#),

## EmulFilter

The incremental main encoder input lines (A, B and Index) are filtered by hardware using the EncFilt parameter.

This filter is performed before these signals are used to count the encoder position by the DSP encoder peripheral.

However, different hardware circuits (at the FPGA) are used to create the encoder emulation outputs. As a result, a different filter mechanism is used for the emulation. This filter is controlled by the EmulFilter parameter.

EmulFilter defines how many input samples (80 MHz) are required to qualify the encoder input for the emulation. An input level is qualified (and transferred to the filter's output) only if it is sampled for EmulFilter consecutive samples.

Description	Value
<b>Mnemonic (keyword):</b>	EmulFilter
<b>CAN code:</b>	403
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	15
<b>Default value:</b>	3
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Encfilt](#)

## EmulIndexType

The EmulIndexType defines how the encoder emulation index output is created.

Two options are supported:

If EmulIndexType == 0:

The encoder emulation index output is aligned (by position and by width) to the encoder emulation encoder output signals.

If EmulIndexType == 1:

The encoder emulation index output is identical to the index of the main encoder (the input to the encoder emulation function).

Description	Value
<b>Mnemonic (keyword):</b>	EmulIndexType
<b>CAN code:</b>	402
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## EmulRat

EmulRat determines the ratio between the main encoder input and the encoder emulation output.

There are 4 types of emulated encoder signals:

1. A – The logical signal of the A side of the digital incremental encoder signal
2. B – The logical signal of the B side of the digital incremental encoder signal
3. Z – The Index signal
4. Pulse – For every edge transition of the emulated encoder signal, a 5 microsecond pulse is generated,

The output for the emulated encoder signals are routed only to differential output signals such that:

- Emulated Encoder A is tied to Differential Output 1
- Emulated Encoder B is tied to Differential Output 2
- Emulated Encoder Z is tied to Differential Output 3
- Emulated Encoder Pulse is tied to Differential Output 4

They can be configured using DOutSelect of that particular output by selecting that output with the value of 1.

### Note:

EmulRat is not supported in Central-i systems. It is not needed because the controller knows the encoder reading of all axes.

Description	Value
<b>Mnemonic (keyword):</b>	EmulRat
<b>CAN code:</b>	69
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	65,536
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If EmulRat = 8, one emulation encoder output pulse will be generated for every 8 incoming pulses.

### See Also:

[DOutPort](#), [DOutSelect](#), [EmulFilter](#).

## EncDir

EncDir can be used to change the direction of the encoder reading. If the encoder reading advances in the positive direction when the motor turns clockwise and EndDir = 0, set EncDir = 1 to cause the reading to advance in the negative direction when the motor turns clockwise.

Description	Value
<b>Mnemonic (keyword):</b>	EncDir
<b>CAN code:</b>	58
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EncFilt

Specifies the digital filter to apply on each of the incremental main encoder input channels (A, B and Index).

The filter is implemented by hardware.

The filter is characterized as follows:

- The input is sampled by the filter mechanism. An input level is qualified only after 6 consecutive samples with identical value. This means that the "1" logic of the signal needs to be sampled at least 6 times and similarly also the "0" logic, to a total of  $(2 * 6)$  samples,
- The filter frequency (sampling frequency) is determined by the EncFilt parameter.
- If  $\text{EncFilt} == 0$ , the filter frequency is equal to the DSP clock. Typically with Agito products: 300MHz.
- If EncFilt is not equal to 0, the filter frequency is:

$$\text{Filter Frequency} = 300 \text{ MHz} / (2 * \text{EncFilt})$$

- The maximal theoretical input frequency, at A and B and Index signals (when no noises are presented) is therefore calculated using the following equation:

If  $\text{EncFilt} = 0$

$$\text{Max Input Frequency} = 300 \text{ MHz} / (2 * 6) \quad \text{MHz}$$

Else

$$\text{Max Input Frequency} = 300 \text{ MHz} / (\text{EncFilt} * 2) / (2 * 6) \quad \text{MHz}$$

Notes:

- The "Max Input Frequency" is the theoretical upper limit of the input frequency as a function of a given EncFilt value. It is calculated assuming ideal square wave signal (infinite slew rate) and ideal electronics (no delays and slew rate at the receiver chips in the controller). The actual maximal frequency that can be used will be somehow smaller, depending on the quality of the signals.
- In addition, if noises are presented in the signal, it will "disturb" the filter from counting 6 consecutive samples of a given logic level. So, again, it is recommended not to push EncFilt too high.
- Bottom line, with a given Max Input Frequency, we recommend setting EncFilt as follows:

$\text{EncFilt} \leq 300 / 24 / \text{Max Input Frequency (MHz)} / 2$   
 $\leq$  means smaller or equal.

The division by 2 is to provide the spares required for none optimal signals and noises.

- The actual value to set for EncFilt at a given system can be initially set according to the above equation but must be carefully tested and adjusted at the system as suitable for the noises that appears on the encoder signals.
- The fastest input signal can be (from the filter point of view): 25MHz (with EncFilt = 0 and assuming no noises).
- The fastest filter frequency is 300 MHz (EncFilt = 0) and the slowest filter frequency is ~0.59 MHz (maximal value for EncFilt is 255).

Description	Value
<b>Mnemonic (keyword):</b>	EncFilt
<b>CAN code:</b>	57
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	255
<b>Default value:</b>	10
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EncRes

EncRes is the number of encoder counts in a mechanical revolution. This number, together with PolePrs is used to determine the pattern of voltage applied to the motor phases. Please use the motor data sheet to enter the correct value of EncRes.

For linear motor enter PolePrs = 1, EncRes = the number of encoder counts per electric cycle

Warning:

If EncRes is wrong unexpected behavior will occur. This can result in severe damage to the controller, motor or any other system parts connected to the motor.

Description	Value
<b>Mnemonic (keyword):</b>	EncRes
<b>CAN code:</b>	56
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,000,000
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[PolePrs](#), [EncType](#)

## EncSubType

Description	Value
<b>Mnemonic (keyword):</b>	EncSubType
<b>CAN code:</b>	610
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EncType

EncType sets the encoder feedback type that is connected to the controlled motor. The values that can be assigned to EncType are listed below. Note that some of these encoder types are not available on all products and some require special options.

Value	Encoder Type
0	Unknown
1	Incremental encoder
2	Sin/Cos
3	Endat
4	SSI
5	Nikon 17 bit encoder

Description	Value
<b>Mnemonic (keyword):</b>	EncType
<b>CAN code:</b>	55
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	6
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EncRes](#),

## EPPDenFract

Description	Value
<b>Mnemonic (keyword):</b>	EPPDenFract
<b>CAN code:</b>	553
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 12
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## EPPDenInteg

Description	Value
<b>Mnemonic (keyword):</b>	EPPDenInteg
<b>CAN code:</b>	552
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 12
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPFiltLength

Description	Value
<b>Mnemonic (keyword):</b>	EPPFiltLength
<b>CAN code:</b>	556
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	12
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPModelRange

Description	Value
<b>Mnemonic (keyword):</b>	EPPModelRange
<b>CAN code:</b>	554
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	800,000
<b>Default value:</b>	90
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPNumFactor

Description	Value
<b>Mnemonic (keyword):</b>	EPPNumFactor
<b>CAN code:</b>	555
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPNumFract

Description	Value
<b>Mnemonic (keyword):</b>	EPPNumFract
<b>CAN code:</b>	551
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 12
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPNumInteg

Description	Value
<b>Mnemonic (keyword):</b>	EPPNumInteg
<b>CAN code:</b>	550
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 12
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPRequest

Description	Value
<b>Mnemonic (keyword):</b>	EPPRequest
<b>CAN code:</b>	559
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EPPState

Description	Value
<b>Mnemonic (keyword):</b>	EPPState
<b>CAN code:</b>	557
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	4
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## ErrLog

---

ErrLog is an array that holds the error log of the controller. When an error occurs two values are saved in ErrLog:

- Error code
- Error time ( in seconds since the last power on)

The first error that occurs is logged in ErrLog[1] and the time of the first error is in ErrLog[2].

Later errors are saved in higher array locations. If there are more than 32 errors the array indexes will start again from 1 and record later errors over the older errors.

The time of the error in ErrLog is in seconds since power on.

The best way to look at ErrLog is by using the PC suite software. PC Suite will translate the error code to text and the time since power up to clock reading according to the PC clock.

The error codes as presented by ErrLog:

1	Message is too long
2	Illegal axis name in the first letter
3	Syntax error
4	Mnemonic too short
5	Closing bracket expected for array index
6	Array index value is invalid
7	Array index value is out of range
8	Assigned value not valid
9	Assigned value out of 32 bits range
10	Invalid character after ']'
11	unexpected character: '=' or '[' expected
12	Mnemonic too long
13	Mnemonic not found in built in keywords table
14	Assigned value out of the range defined for this keyword
15	The number of received bytes is unexpected
16	Header must be 0
17	Recorded data not valid after power up
18	Still recording, cannot upload or start recording
19	Cannot start recording of 0 or negative length
20	Index is larger than max allowed index for this keyword
21	The parameter assignment or function call are not valid during motion
22	The parameter assignment or function call are not valid with motor on
23	Trying to assign value to a read only parameter
24	This keyword requires array index
25	This keyword does not require an array index
26	During save to flash could not unlock
27	Could not erase flash
28	Error writing to flash
29	Flash full, not all parameters were saved
30	Flash checksum verification during load failed
31	Can't turn motor on if commutation was not found yet
32	Unexpected data encountered while loading from flash
33	Can't start recording because one or more of the requested parameters is a function
34	Number of parameters in RecLength exceeds the recording array size
35	RecTrigMask = 0 will not enable recording to begin
36	RecStop was activated before a recording was started
37	RecTrigSrc is a function
38	RecStop stopped recording before trigger. Data incomplete
39	Can't start motion if motor is off
40	Motion mode not supported yet by this firmware version

41	Can't begin motion when already in motion
42	A line checksum error encountered during firmware download
43	An unexpected character was received during firmware download
44	Wrong number of characters in download file line
45	An address within the download file falls out of range
46	Wrong password entered. Download firmware process aborted
47	Timeout during firmware download process
48	Trying to begin motion with invalid motion mode value
49	Wrong index in recording parameter
50	This function keyword requires a parameter
51	This function keyword does not require a parameter
52	The numeric stack of the user program is full
53	Trying to pop from an empty stack. Please contact agito to debug!
54	Pointer to numeric stack is out of range
55	The user program thread indicated is out of range
56	Math function was called without enough data in stack
57	Math function called with divisor 0
58	The result of the math function exceeds the allowed numeric range
59	Trying to access a location beyond the full area of the stack
60	Math operation requested does not exist
61	Negative power not implemented yet
62	Negative value entered to square root operation
63	MATH function not yet implemented
64	Recording trigger source can't be a function
65	Recording parameter from a non valid axis
66	Recording parameter with CAN code out of range
67	Recording gap out of range
68	Recording trigger source from a non valid axis
69	Recording trigger source with a CAN code out of range
70	Wrong index in recording trigger source
71	Recording trigger position out of range
72	Recording trigger type out of range

Description	Value
<b>Mnemonic (keyword):</b>	ErrLog
<b>CAN code:</b>	235
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 384
<b>Save to flash:</b>	No

<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EthernetIP

The IP address for Ethernet communication.

Description	Value
<b>Mnemonic (keyword):</b>	EthernetIP
<b>CAN code:</b>	600
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	255
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EthernetMAC](#), [EthernetPort](#),

## EthernetMAC

The MAC address for Ethernet communication.

Description	Value
<b>Mnemonic (keyword):</b>	EthernetMAC
<b>CAN code:</b>	602
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 6
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	255
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EthernetIP](#), [EthernetPort](#),

## EthernetPort

The port number of Ethernet communication.

Description	Value
<b>Mnemonic (keyword):</b>	EthernetPort
<b>CAN code:</b>	601
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	65,535
<b>Default value:</b>	50,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[EthernetIP](#), [EthernetMAC](#),

## EventBegPos

EventBegPos is the position of the first event that will be generated in modes 1 and 2 (single event or by gap).

Description	Value
<b>Mnemonic (keyword):</b>	EventBegPos
<b>CAN code:</b>	181
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EventOn](#), [EventGap](#), [EventEndPos](#), [EventCntr](#), [EventPulseWid](#), [EventType](#), ETStart, ETEnd



## EventCntr

EventCntr count the events that occurred since the last EventOn. Toggling EventOn will reset the counter. This counter can also be reset by the user.

Description	Value
<b>Mnemonic (keyword):</b>	EventCntr
<b>CAN code:</b>	186
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[EventOn](#), [EventBegPos](#), [EventEndPos](#), [EventGap](#), [EventPulseWid](#)

## EventEndPos

EventEndPos is the highest position for which events will be generated when using EventType = 2 (by gap). EventEndPos does not have to be a location where an event is generated.

Example:

EventType = 2 // Event generation by gap

EventBegPos = 1000

EventGap = 2000

EventEndPos = 8000

EventOn = 1 // This assignment must be made in a position that is smaller than EventBegPos to prevent unexpected behavior.

This sequence will cause the event out to be "ON" for the duration defined by PulseWidth when passing in the following positions: 1000, 3000, 5000, 7000.

After passing position 8000 no more events are generated, and EventOn should be toggled to restart generating events.

Description	Value
<b>Mnemonic (keyword):</b>	EventEndPos
<b>CAN code:</b>	183
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[EventOn](#), [EventGap](#), [EventBegPos](#), [EventCntr](#), [EventPulseWid](#)

## EventGap

EventGap defines the position gap between event generations. Note that if Event Gap is small, and the velocity is high, a high PulseWidth may cause events to overlap.

Description	Value
<b>Mnemonic (keyword):</b>	EventGap
<b>CAN code:</b>	182
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[EventOn](#), [EventBegPos](#), [EventNextPos](#), [EventCntnr](#), [EventPulseWid](#), [EventEndPos](#)

## EventNextPos

Description	Value
<b>Mnemonic (keyword):</b>	EventNextPos
<b>CAN code:</b>	319
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[EventOn](#), [EventBegPos](#), [EventGap](#), [EventCntr](#), [EventPulseWid](#), [EventEndPos](#)

## EventOn

EventOn = 1 will load the first position according to the EventType to the position comparator and turn on the events. EventOn = 1 should be entered when the motor is in a position that is smaller than the first requested event to prevent unexpected behavior.

Event and Lock are mutually exclusive functionalities. EventOn = 1 will automatically cause LockEN to become 0.

Description	Value
<b>Mnemonic (keyword):</b>	EventOn
<b>CAN code:</b>	178
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EventBegPos](#), [EventGap](#), [EventEndPos](#), [EventCntr](#), [EventPulseWid](#), [EventType](#), ETStart, ETEnd

## EventPulseWid

Description	Value
<b>Mnemonic (keyword):</b>	EventPulseWid
<b>CAN code:</b>	179
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-10,000
<b>Max value:</b>	10,000
<b>Default value:</b>	50
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EventSelect

Description	Value
<b>Mnemonic (keyword):</b>	EventSelect
<b>CAN code:</b>	317
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	7
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EventTable

Description	Value
<b>Mnemonic (keyword):</b>	EventTable
<b>CAN code:</b>	316
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 150
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## EventTableBeg

Description	Value
<b>Mnemonic (keyword):</b>	EventTableBeg
<b>CAN code:</b>	184
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	150
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EventTableEnd

Description	Value
<b>Mnemonic (keyword):</b>	EventTableEnd
<b>CAN code:</b>	185
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	150
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EventTableSel

Description	Value
<b>Mnemonic (keyword):</b>	EventTableSel
<b>CAN code:</b>	318
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 150
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	7
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## EventType

Events are pulses on a designated output that are generated when the actual feedback position equals a required compare position. EventType determines different compare options:

EventType = 0: A pulse will be generated when the feedback position equals the position in EventBegPos

EventType = 1: Event by gap. A first pulse is generated when the position equals EventBegPos. Another pulse is generated every time the distance defined by EventGap is passed. When the position exceeds EventEndPos pulses stop being generated.

EventType = 2: Events by table. A table of positions where events should be generated is entered into GenData[] array. EventTableBeg is the index of the event table start. EventTableEnd is the index of the event table end. The position in the table must be ordered from low to high.

Description	Value
<b>Mnemonic (keyword):</b>	EventType
<b>CAN code:</b>	180
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[EventOn](#), [EventGap](#), [EventEndPos](#), [EventCntr](#), [EventPulseWid](#), [EventTableBeg](#), [EventTableEnd](#).

## FastIdDownSam

Description	Value
<b>Mnemonic (keyword):</b>	FastIdDownSam
<b>CAN code:</b>	541
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3
<b>Default value:</b>	3
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## FastIdInit

Description	Value
<b>Mnemonic (keyword):</b>	FastIdInit
<b>CAN code:</b>	540
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## FIFOClear

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOClear
<b>CAN code:</b>	290
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## FIFOCycleTime

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOCycleTime
<b>CAN code:</b>	283
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	16,384,000
<b>Default value:</b>	16,384
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#),



## FIFOPushCycle

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOPushCycle
<b>CAN code:</b>	284
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	16,384,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#),

## FIFOPushLinP

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOPushLinP
<b>CAN code:</b>	285
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#),

## FIFOPushLinV

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOPushLinV
<b>CAN code:</b>	286
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-60,000,000
<b>Max value:</b>	60,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#),

## FIFOPushParA

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOPushParA
<b>CAN code:</b>	288
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,000,000,000
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#),

## FIFOPushParP

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOPushParP
<b>CAN code:</b>	287
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#),

## FIFORemove

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFORemove
<b>CAN code:</b>	289
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#)

## FIFOStatus

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOStatus
<b>CAN code:</b>	282
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#)

## FIFOType

---

FIFO is a special motion mode in which the controller performs a sequence of linear and parabolic segments, as defined by the user before the motion and optionally also during the motion.

The motion is created according to motion segments which are stored in a FIFO memory.

The motion segments definitions can be pushed to the FIFO at any time (before or during the motion), providing that the FIFO is not full. If the FIFO is full, the push operation is rejected with a suitable error.

If, during a motion in this mode, the controller reaches the last element in the FIFO, and completing this motion segment, and yet no new element was pushed, the motion is automatically ended.

The motion can be also stopped using the Stop or the StopFIFO functions. The first decelerate to zero speed and the second makes the currently executed motion segment to be the last segment.

Each motion segment can be of type Velocity or Acceleration. Velocity type segment is a segment in which the velocity reference is constant and Acceleration type segment is a segment in which the acceleration reference is constant.

The time length of each segment (FIFOCycleTime) is a fixed value of number of control samples. However, it can be modified at any time that the controller is ending a given segment and starting a new one.

The FIFO is of size 512 entries. Each entry has type and value. See below for possible FIFO entry types. If all entries are motion entries, the FIFO can hold up to 512 motion segments. Of course, it can be re-filled over the communication, during the motion sequence.

The controller provides variety of parameters and functions to handle the FIFO and the motion behavior, as described in details below.

### Possible segment motions

---

#### 1. Velocity type motion segment:

- a. Segment duration is known in number of samples (sample time of the control loop). It is stored in the parameter FIFOCycleTime.
- b. The segment starts naturally from the last position reference.
- c. Motion definition can be given by one of:
  - i. Move linearly with given delta for position reference.
    1. Final target position is calculated using the given delta. The delta is given with scaling of the controller SAMPLING\_FREQUENCY (typically 16384 Hz). This means that a delta of 1 [count] is



entered as 16384. And a delta of 2.5 [counts] is entered as  $2.5 * 16384 = 40960$ .

This means that fractional values are supported to ensure trajectory smoothness between the segments.  
As a result, the delta is limited to  $\pm 131071$ .

If larger delta is required, the segment shall be divided into few segments, or another motion definition shall be used (see below).

2. Segment velocity is calculated, in [counts/sec].
3. The velocity is used to increment the position reference at each sample.
4. At last sample, the position reference is assigned to the desired value (to compensate for possible increments inaccuracies).

ii. Move linearly with given velocity.

1. Segment velocity is provided in [counts/sec].
2. The velocity is used to increment the position reference at each sample.
3. At last sample, there is the resulted position reference, as calculated by the increments.
4. Note that in this definition type, the segment is not limited in length as the previous one (having limited value of the position delta).

2. Acceleration type motion segment:

(Details explanations are not repeated as they are similar to the above described cases)

- a. Segment duration is known in number of samples.
- b. The segment starts naturally from the last position reference, as well as from the end velocity of the previous segment.
- c. Motion definition can be given by one of:

i. Move parabolic with given delta for position reference.

1. Final target position is calculated using the given delta.
2. Segment acceleration (or deceleration) is calculated, in [counts/sec<sup>2</sup>].
3. The acceleration is used to increment the desired velocity and position at each sample, based on:

$$VelRefProfiler = VelRefProfiler + Acceleration * Ts$$

$$PosRef = PosRef + VelRef$$

Ts is the controller sampling time, typically 1/16384.

4. At last sample, the position reference is assigned to the desired value (to compensate for increments inaccuracies).

- ii. Move parabolic with given acceleration.
  1. Segment acceleration (or deceleration, if negative) is provided in [counts/sec<sup>2</sup>].
  2. The acceleration is used to increment the desired velocity and position at each sample, based on:

$$\begin{aligned} \text{VelRefProfiler} &= \text{VelRefProfiler} + \text{Acceleration} * T_s \\ \text{PosRef} &= \text{PosRef} + \text{VelRef} * T_s \end{aligned}$$

$T_s$  is the controller sampling time, typically 1/16384.

3. At last sample, there is the resulted velocity reference and position reference, as calculated by the increments.

## FIFO entry types

1. FIFO Motion Mode parameters:
  - a. The FIFO motion mode performs segment motions as a function of the segment definition itself, and as a function of some parameters (like the FIFOCycleTime – the time duration of a single segment motion).
  - b. The values of these parameters can be modified by pushing a new value into the FIFO. This value is used when the controller reaches this location in the FIFO. It is always between two motion segments. As a result, it is a way to assign values to the relevant parameters in a synchronized way.
  - c. The new value is effective immediately and will be used till a new value is reached in the FIFO.
  - d. Such FIFO entries do not consume motion time.
  - e. A given FIFO motion sequence can use new parameters between each segment, or can avoid using new parameters at all. Pushing new motion mode parameters into the FIFO is optional and shall be used according to the desired overall motion sequence/contour.
  - f. It is assumed that there will be limited number of such entries between motion segments (to avoid CPU load within the control interrupt when reaching this location in the FIFO).
  - g. FIFO entry type to assign a value to a parameter is given in one of:
    - i. FIFO\_CYCLE\_TIME:  
It assigns a value to the parameter FIFOCycleTime. This is the FIFO cycle time, which is the duration, in control samples, of each FIFO motion segment.

Use the function FIFOPushCycle to make such a push:

For example:

FIFOPushCycle, 1000

Pushes a value of 1000 into the FIFO, with type of FIFO\_CYCLE\_TIME.

It will assign the value of 1000 to FIFOCycleTime, when the FIFO motion will reach this entry in the FIFO. It then will be used for all coming segments, till it will be changed again.

ii. More to be added in the future.

## 2. FIFO Motion segments:

- a. The FIFO motion mode performs segment motions that are located in the FIFO array.
- b. A motion segment is created by pushing a motion segment into the FIFO.
- c. This segment will be executed when the FIFO motion will reach this entry.
- d. Each motion segment “consumes” a time period which is equal to the value of FIFOCycleTime (its value when this segment is executed).
- e. Various type of motion segments are supported, as defined above in details.
- f. FIFO entry type to define a motion segment is given in one of:

### 1. LINEAR\_BY\_DELTAPOS:

It is a FIFO entry that initiates a segment of velocity motion, defined by the delta position reference from the start of the segment to its end.

The value is scaled by the sampling frequency of the control loop. Naturally with Agito controllers it is 16384 [Hz].

Use the function FIFOPushLinP to make such a push:

For example:

FIFOPushLinP, 163024896

Pushes a value of 163024896 into the FIFO, with a type of LINEAR\_BY\_DELTAPOS.

It will initiate a motion segment of linear motion, with delta position of 9950.25 ( $=163024896/16384$ ).

### 2. LINEAR\_BY\_VELOCITY:

Similarly to the above, but the relevant pushing function is FIFOPushLinV, the pushed type is LINEAR\_BY\_VELOCITY and the pushed value is the velocity for this segment, in [counts/sec] (no scaling is required).

### 3. PARABOLIC\_BY\_DELTAPOS:

It is a FIFO entry that initiates a segment of acceleration motion, defined by the delta position reference from the start of the segment to its end.

The value is scaled by the sampling frequency of the control loop. Naturally with Agito controllers it is 16384 [Hz].

Use the function `FIFOPushParP` to make such a push:

For example:

`FIFOPushParP, 163024896`

Pushes a value of 163024896 into the FIFO, with a type of `PARABOLIC_BY_DELTAPOS`.

It will initiate a motion segment of acceleration (parabolic) motion, with delta position of 9950.25 ( $=163024896/16384$ ).

4. `PARABOLIC_BY_ACCLERATION`:

Similarly to the above, but the relevant pushing function is `FIFOPushParA`, the pushed type is `PARABOLIC_BY_ACCLERATION` and the pushed value is the acceleration for this segment, in [counts/sec<sup>2</sup>] (no scaling is required).

## FIFO parameters and functions

---

The following are the keywords relevant for FIFO motions:

1. `FIFOValue[]`

This keyword shows the FIFO as an array. It shows the “value” entries. Each element in the array is a part of a FIFO entry. The complete FIFO entry consists of this value and of the type of this value (see next item).

2. `FIFOType[]`

This keyword shows the FIFO as an array. It shows the “type” entries. Each element in the array is a part of a FIFO entry. The complete FIFO entry consists of a value (see previous item) and of the type of this value.

3. `FIFOStatus[]`:

- a. `FIFOStatus[1]` → `FIFO_INDEX`  
Holds the current value of the pointer into the FIFO.
- b. `FIFOStatus[2]` → `FIFO_FREE`  
Holds the number of free entries in the FIFO.
- c. `FIFOStatus[3]` → `FIFO_COUNT_DOWN`  
Holds the number of [samples] remained to complete the current motion segment.
- d. `FIFOStatus[4]` → `FIFO_SEGMENT_VELOCITY`  
Holds the velocity that is used in this segment, in [counts/sec].

- e. FIFOStatus[5] → FIFO\_SEGMENT\_ACCELERATION  
Holds the acceleration that is used in this segment, in [counts/sec<sup>2</sup>].
- 4. Push functions:
  - a. FIFOPushCycle  
Push a value into the FIFO that will be used as a new value for FIFOCycleTime.
  - b. FIFOPushLinP  
Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Velocity (linear position) and the value defines the delta position (in [counts] and scaled by the control sampling frequency) for this segment (from its start till its end).
  - c. FIFOPushLinV  
Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Velocity (linear position) and the value defines the velocity (in [counts/sec]) during this segment.
  - d. FIFOPushParP  
Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Acceleration (parabolic position) and the value defines the delta position (in [counts] and scaled by the control sampling frequency) for this segment (from its start till its end).
  - e. FIFOPushParA  
Push a value into the FIFO that will be used as a motion segment instruction. The motion segment is of type Acceleration (parabolic position) and the value defines the acceleration (in [counts/sec<sup>2</sup>]) during this segment.
- 5. FIFORemove  
Removes the last entry (recently pushed) from the FIFO.
- 6. FIFOClear  
Clears the entire FIFO.  
  
Note:  
If a FIFO motion halts due to some fault or limit, its FIFO may be not empty (it will remain as it was when the motion was halted). You may need to perform FIFOClear before starting a new FIFO motion.
- 7. FIFOCycleTime  
Defines the period of time, in [samples], for a motion segment.
- 8. MotionMode  
Needs to have the relevant value to define FIFO motion (MotionMode = 9).
- 9. Begin  
Starts a FIFO motion sequence if the MotionMode is set to this mode (9).

10. Stop  
Stops the motion using deceleration.
11. StopFIFO  
Stops the motion by defining the currently executed motion segment as the last segment.

#### Miscellaneous

---

1. When the motion is ended (by StopFIFO or by ending the last segment), the remaining position reference's fraction (if any) is cleared to zero.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOType
<b>CAN code:</b>	281
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 128
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	5
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## FIFOValue

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	FIFOValue
<b>CAN code:</b>	280
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 128
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#)

## Force

Description	Value
<b>Mnemonic (keyword):</b>	Force
<b>CAN code:</b>	582
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## ForceAlnTh

Description	Value
<b>Mnemonic (keyword):</b>	ForceAlnTh
<b>CAN code:</b>	584
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-100,000
<b>Max value:</b>	100,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrAlnTh](#),

## ForceCmdCntr

Description	Value
<b>Mnemonic (keyword):</b>	ForceCmdCntr
<b>CAN code:</b>	574
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	122,070,306
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceCmdHTime

Description	Value
<b>Mnemonic (keyword):</b>	ForceCmdHTime
<b>CAN code:</b>	572
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-122,070,306
<b>Max value:</b>	122,070,306
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceCmdIndex

Description	Value
<b>Mnemonic (keyword):</b>	ForceCmdIndex
<b>CAN code:</b>	573
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	20
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceCmdSlope

Description	Value
<b>Mnemonic (keyword):</b>	ForceCmdSlope
<b>CAN code:</b>	569
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceCmdSrc

The ForceCmdSrc is used to choose source of Force; so far we have three types of force command source, which are analog input, scheduled force command, and interpolated scheduled force command.

Description	Value
<b>Mnemonic (keyword):</b>	ForceCmdSrc
<b>CAN code:</b>	570
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ForceCmdVal](#)

## ForceCmdVal

The ForceCmdVal is used to set the value of force command, if the force command source is not analog input;

This is array, to set force command value at different stage, max is 20.

Description	Value
<b>Mnemonic (keyword):</b>	ForceCmdVal
<b>CAN code:</b>	571
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-16,000
<b>Max value:</b>	16,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ForceCmdSrc](#)

## ForceErr

Description	Value
<b>Mnemonic (keyword):</b>	ForceErr
<b>CAN code:</b>	583
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## ForceFFW

Description	Value
<b>Mnemonic (keyword):</b>	ForceFFW
<b>CAN code:</b>	589
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceGain

Description	Value
<b>Mnemonic (keyword):</b>	ForceGain
<b>CAN code:</b>	577
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceKd

Description	Value
<b>Mnemonic (keyword):</b>	ForceKd
<b>CAN code:</b>	588
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceKi

Description	Value
<b>Mnemonic (keyword):</b>	ForceKi
<b>CAN code:</b>	578
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForcePosErrTh

Description	Value
<b>Mnemonic (keyword):</b>	ForcePosErrTh
<b>CAN code:</b>	576
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-327,680
<b>Max value:</b>	327,680
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceRef

Description	Value
<b>Mnemonic (keyword):</b>	ForceRef
<b>CAN code:</b>	581
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceRefFilt

Description	Value
<b>Mnemonic (keyword):</b>	ForceRefFilt
<b>CAN code:</b>	586
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	500,000
<b>Default value:</b>	10,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ForceRefFOn

Description	Value
<b>Mnemonic (keyword):</b>	ForceRefFOn
<b>CAN code:</b>	579
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## ForceVelFFW

Description	Value
<b>Mnemonic (keyword):</b>	ForceVelFFW
<b>CAN code:</b>	580
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## FrictionComp

Friction compensation is activated if the user variable FrictionComp is not zero.

This parameter has always positive values.

When motion starts (Begin command) a flag is set. Then, when VelProfiler is not zero, this value is added to the velocity control integral and the flag is cleared.

Positive value is added if VelProfiler is positive.

Negative value is added if VelProfiler is negative.

The FrictionComp is in [mA].

This function is valid only for motion modes that use VelProfiler.

Description	Value
<b>Mnemonic (keyword):</b>	FrictionComp
<b>CAN code:</b>	406
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	5,000
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## FwdPLim

FWDPLim is the forward position limit in user units. This is the maximum allowed position for the motor. If the position exceeds this limit the motor is stopped. It will not be possible to start a new motion in the positive direction. Only negative motion is allowed until the position returns into the limits.

Description	Value
<b>Mnemonic (keyword):</b>	FwdPLim
<b>CAN code:</b>	83
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	2,000,000,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[RevPLim](#),

## GantryAccFFW

Description	Value
<b>Mnemonic (keyword):</b>	GantryAccFFW
<b>CAN code:</b>	655
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	500,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## GantryFdbk

The GantryFdbk parameter is a read-only parameter. It provides the MIMO gantry control feedbacks.

AGantryFdbk is equal to  $(APos + BPos) / 2$ .

BGantryFdbk is equal to  $(APos - BPos)$ .

Note that these parameters are calculated even when the Gantry mode is disabled.

Note that the above calculation considers also the GantryOffset, as explained below, although not written in the above equations, for simplicity.

?GantryFdbk with "?" not equal to "A" or "B" has no use and will always returns a value of 0.

Description	Value
<b>Mnemonic (keyword):</b>	GantryFdbk
<b>CAN code:</b>	652
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

## GantryOffset

The GantryOffset is a read-only parameter.

AGantryOffset is calculated once when AGantryOn is switched by the user from 0 to 1.

$AGantryOffset = APosRef - BPosRef$ .

The GantryOffset is later used for the calculation of the GantryFdbk readings, so that the initial offset between the readings of the two encoders is ignored.

The actual calculation is:

$AGantryFdbk$  is equal to  $(APos + BPos + AGantryOffset) / 2$

$BGantryFdbk$  is equal to  $(APos - BPos - AGantryOffset)$

?GantryOffset with "?" not equal to "A" has no use and will always returns a value of 0.

Description	Value
<b>Mnemonic (keyword):</b>	GantryOffset
<b>CAN code:</b>	653
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[GantryAccFFW](#), [GantryFdbk](#), [GantryOn](#), [GantryPosGain](#), [GantryVelGain](#), [GantryVelKi](#),  
[GantryYawRef](#),

## GantryOn

AGantryOn control the operation of the gantry mode. With AGantryOn=0, the gantry mode is disabled and each axis can be moved and controlled independently. With AGantryOn=1, the gantry mode is enabled and the control scheme is automatically changed to gantry MIMO control.

When Gantry is on, motions to the gantry stage is done by commanding motions to the A axis.

Note that ?GantryOn, with "?" not equal to "A" will result with an error. Gantry can be activated only on A axis, and A and B axes are to be always used for the Gantry control.

Note that AGantryOn is automatically cleared to 0 when AMotorOn or BMotorOn are 0. The Gantry mode is typically enabled by the user only after both A and B motors are enabled.

Description	Value
<b>Mnemonic (keyword):</b>	GantryOn
<b>CAN code:</b>	650
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[GantryAccFFW](#), [GantryFdbk](#), [GantryOffset](#), [GantryPosGain](#), [GantryVelGain](#), [GantryVelKi](#),  
[GantryYawRef](#),

## GantryPosGain

Gantry Mode Position Loop Proportional Gain.

Description	Value
<b>Mnemonic (keyword):</b>	GantryPosGain
<b>CAN code:</b>	654
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	100,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[GantryAccFFW](#), [GantryFdbk](#), [GantryOffset](#), [GantryOn](#), [GantryVelGain](#), [GantryVelKi](#),  
[GantryYawRef](#),



## GantryVelGain

Gantry Mode Velocity Loop Proportional Gain.

Description	Value
<b>Mnemonic (keyword):</b>	GantryVelGain
<b>CAN code:</b>	656
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	100,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[GantryAccFFW](#), [GantryFdbk](#), [GantryOffset](#), [GantryOn](#), [GantryPosGain](#), [GantryVelKi](#), [GantryYawRef](#),

## GantryVelKi

Gantry Mode Position Loop Integral Gain.

Description	Value
<b>Mnemonic (keyword):</b>	GantryVelKi
<b>CAN code:</b>	657
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	100,000
<b>Default value:</b>	100
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[GantryAccFFW](#), [GantryFdbk](#), [GantryOffset](#), [GantryOn](#), [GantryPosGain](#), [GantryVelGain](#),  
[GantryYawRef](#),

## GantryYawRef

Description	Value
<b>Mnemonic (keyword):</b>	GantryYawRef
<b>CAN code:</b>	651
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-20,000
<b>Max value:</b>	20,000
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[GantryAccFFW](#), [GantryFdbk](#), [GantryOffset](#), [GantryOn](#), [GantryPosGain](#), [GantryVelGain](#),  
[GantryVelKi](#),

## GenData

GenData is an array assigned for user general data. In this array the user can store any data. It is usually used for tables such as ECAM tables.

Description	Value
<b>Mnemonic (keyword):</b>	GenData
<b>CAN code:</b>	237
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 1000
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

AGenData[900]=10

**See Also:**

## GoToCurrMode

Change to Current control mode directly.

Change the operation mode to Current Operation Mode, and do everything that may be required, like halting motion, clearing counters and pointers (for time based Current Control Mode) before really changing OperationMode.

Description	Value
<b>Mnemonic (keyword):</b>	GoToCurrMode
<b>CAN code:</b>	335
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CurrPosErrTh](#), [CurrAlnTh](#),

## GoToForceMode

Change to Force control mode directly.

Change the operation mode to Force Operation Mode, and do everything that may be required, like halting motion, clearing counters and pointers (for time based Force Control Mode) before really changing OperationMode.

Description	Value
<b>Mnemonic (keyword):</b>	GoToForceMode
<b>CAN code:</b>	575
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ForceInTh](#),

## GoToPosMode

Change to Force control mode directly.

We switch back to position mode, no motion, PosRef is the last position.

Nothing is done if we are already in Position Mode.

Error is returned if we are in Velocity Mode (as this mode is not handled at the moment, can be added in the future if will be needed... at the moment, in Velocity Operation Mode, we do not prepare all Position variables to be ready for the switch back ....).

Description	Value
<b>Mnemonic (keyword):</b>	GoToPosMode
<b>CAN code:</b>	336
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[GoToCurrMode](#), [GoToForceMode](#),

## HallsAngle

Description	Value
<b>Mnemonic (keyword):</b>	HallsAngle
<b>CAN code:</b>	384
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 6
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	360
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[HallsValue](#),



## HallsValue

Description	Value
<b>Mnemonic (keyword):</b>	HallsValue
<b>CAN code:</b>	383
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	6
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[HallsAngle](#),

## HomeStat

Description	Value
<b>Mnemonic (keyword):</b>	HomeStat
<b>CAN code:</b>	111
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[HomingOn](#),

## HomingDef

Please refer to the [HomingOn](#) keyword page for detailed description of the Homing function at Agito's controllers.

Description	Value
<b>Mnemonic (keyword):</b>	HomingDef
<b>CAN code:</b>	341
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 150
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[HomingOn](#),

## HomingOn

---

Agito's controllers support a built-in homing process that can be "programmed" by the user to create any homing process that is optimally suitable for the application.

The Homing process is controlled by two parameters: the HomingOn parameter and the HomingDef[] array parameter.

The status of the Homing process is reported at the HomingStat parameter.

The HomingOn parameter is cleared to "0" upon power on or reset. Once it is set to "1" by the user, the controller will start the homing process according to the definitions at the HomingDef[] array parameter, while reporting its status, during the process, at the HomingStat parameter.

The HomingOn parameter is cleared by the controller upon completion of the homing process (whether successfully or due to some error, as described below).

The Homing process consists of steps. The number of steps and the action to perform at each step (move to limit, move to index, set position...) are defined by the values at the HomingDef[] array parameter, as described in details below.

Upon successful completion of the last step, the homing is completed and the HomingOn parameter is cleared by the controller.

At each step, some errors can occur (depending on the step type/action). In case of an error, the homing process will be aborted, HomingOn will be cleared, and the HomingStat will report a suitable value, as listed below.

The maximal number of homing process steps is the size of the HomingDef[] array, divided by 10. Typically at Agito's controllers, the size of HomingDef[] is 150, which means that the user can create an homing process that includes up to 15 steps.

The HomingStat may report one of the following statuses:

HomingStat value	Meaning
0	No homing was done after power on or reset
Positive value (not 100)	Homing is in process. HomingStat value reflects the number of the currently processed step in the homing process.
-1	The homing process failed (and aborted) due to wrong parameter at HomingDef[] array parameter (the parameters related to each step are checked when each step starts during the homing process).
-2	The homing process failed (and aborted) due to timeout during one of the homing steps. Each step (if relevant) is defined with a timeout. In case this timeout pass and the step has not yet completed, it is an error.
-3	The homing process failed (and aborted) due to unexpected motor off. This means that during one of the homing steps, the axis was disabled due to some fault (reflected at the value of ConFlt) and the step could not be completed.
-4	The homing process failed (and aborted) due to wrong motion reason. This means that a given step at the homing process, which expects a given reason for end of motion (RLS, index, reached target...) encounters a different reason for ending the motion.
-5	The homing process failed (and aborted) due to wrong step type. This means that the homing process reached a step whose type (as defined in the HomingDef[] array parameter) is not recognized.
-6	The homing process failed (and aborted) due to in-motion. This means that the homing process detected that the axis is in-motion, while it is starting a new step.
-7	The homing process failed (and aborted) due to too many steps. This error will happen if the homing process will reach the last step defined in the HomingDef[] array parameter, and it will not be an end of homing step.
-8	The homing process failed (and aborted) due to unexpected limit. This error is relevant only to one of the homing steps type: HOMING_TYPE_CHECK_THAT_OUT_OF_LIMITS. See more details below.
100	The homing process has been successfully completed.

As explained above, the HomingDef[] array parameter is used to define the homing process, by definitions of the homing steps and the type and parameters for each step.

HomingDef[1] defines the type of the first step.

HomingDef[2-10] defines the parameters for this step.

The number of actually used parameters, and the meaning of each parameter, depend on the step type.

Similarly, HomingDef[11-20] defines that 2<sup>nd</sup> step. HomingDef[21-30] defines the 2<sup>nd</sup> step.

It is extremely convenience to define the homing process parameters at a text file (\*.par, for parameters file) that can be downloaded (using the PC Suite) to controller to define the homing process. Here is an example for such file (definition of homing process for a given application), which will be followed by a detailed description of the supported step types and the parameters for each type:

```
//  
// 1. Go fast into reverse limit  
//  
AHomingDef[1]=1  
AHomingDef[2]=-10000  
AHomingDef[3]=10000000  
AHomingDef[4]=10000000  
AHomingDef[5]=163840  
//  
// 2. Wait 100ms  
//  
AHomingDef[11]=7  
AHomingDef[12]=1638  
//  
// 3. Jog forward slow to index  
//  
AHomingDef[21]=4  
AHomingDef[22]=1000  
AHomingDef[23]=1000000  
AHomingDef[24]=1000000  
AHomingDef[25]=163840  
//  
// 4. Wait 100ms  
//  
AHomingDef[31]=7  
AHomingDef[32]=1638  
//  
// 5. Set position to 0  
//  
AHomingDef[41]=6  
AHomingDef[42]=0  
//  
// 6. End of homing  
//  
AHomingDef[51]=0
```

Users can also use the Homing tool (window) of the PC Suite which provides easy configuration of the Homing process and parameters, as well as built-in homing scenarios.

Now it is needed to define the details of the HomingDef[] content, which defines the homing process:

The description below will refer to HomingDef[1-10]. However, the same is relevant for HomingDef[11-20] (2<sup>nd</sup> step) and so on.

HomingDef[1] defines the type of the step. The following table shows the available types:

HomingDef[1] (or [11], [21] ...) value	Step type
0	End homing. This must be the last step.
1	Jog (jogging) into limit.
2	Check that out of limits
3	Move relative PTP (point to point)
4	Jog to index
5	Move to index position
6	Set position (***)
7	Wait N samples
8	Enable (or disable) the motor
9	Move to hard stop (detected by motor stuck) (***)
10	Move to hard stop (detected by high position error) (***)
11	Jog to a change in the Home discrete input
12	Move absolute PTP (point to point)
13	Set position software limits (RevPLim and FwdPLim parameters)

(\*\*\*) These homing step types have some limitations regarding position value setting. Please read the description below carefully.

The following table presents the description for each type, and its parameters:

Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
End homing. This must be the last step.	The homing process will stop. Reaching this step means that the homing is completed successfully.	No parameters

Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
Jog (jogging) into limit.	Move in jogging mode, in the speed and direction as defined at the step parameters. Successfully completed when the motion stops due to the relevant limit switch.	<p>HomingDef[2]: Jogging speed. The sign is the direction and also defines which limit switch to look for.</p> <p>HomingDef[3]: Acceleration/Deceleration.</p> <p>HomingDef[4]: Emergency deceleration.</p> <p>HomingDef[5]: Maximum step time, in [samples]</p>
Check that out of limits	Check if both RLS and FLS are not activated. If one of them is activated, terminate the homing process with a suitable error.	No parameters
Move relative PTP (point to point)	Move with the provided motion parameters to a given relative distance.	<p>HomingDef[2]: Maximum speed.</p> <p>HomingDef[3]: Acceleration/Deceleration.</p> <p>HomingDef[4]: Relative distance (can be positive or negative).</p> <p>HomingDef[5]: Maximum step time, in [samples]</p>



Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
Jog to index	Jog with the provided motion parameters till an index is detected. The jogging speed shall be low enough to ensure that the index is detected. Theoretically, low enough should be smaller than 16384 counts/sec. Recommended values are smaller than 8000 counts/sec.	HomingDef[2]: Jogging speed. The sign is the direction of motion.  HomingDef[3]: Acceleration/Deceleration.  HomingDef[4]: Emergency deceleration.  HomingDef[5]: Maximum step time, in [samples]
Move to index position	Move (using the provided motion parameters) to the last recorded index position.	HomingDef[2]: Maximum speed.  HomingDef[3]: Acceleration/Deceleration.  HomingDef[4]: Emergency deceleration.  HomingDef[5]: Maximum step time, in [samples]
Set position	Set the current position to the provided value.  Note that this step will do nothing in case the conditions for SetPosition are not met. Refer to the SetPosition keyword manual page.	HomingDef[2]: Desired position value to set at the current position.
Wait N samples	Wait N (as provided by the step parameters) samples.	HomingDef[2]: Number of cycle to wait before advancing to the next step.

Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
Enable (or disable) the motor	Enables or disables the motor	<p>HomingDef[2]: 0 to disable the motor. 1 to enable the motor.</p> <p>HomingDef[3]: Maximum step time, in [samples]</p>
Move to hard stop (detected by motor stuck)	<p>Move (using the provided motion parameters) till hard stop is detected using the provided parameters to detect motor stuck.</p> <p>Motor stuck is detected when the motor velocity (absolute value) is lower than the given velocity threshold and the motor current (absolute value) is higher than the given motor current threshold, both for consecutive period of the given stuck time.</p> <p>Once motor stuck is detected, the motor position is set to the given "position to set" value.</p> <p>The sign of the Maximum speed parameter defines the direction of the motion.</p> <p>Note that this step will not set the desired position in case the conditions for SetPosition are not met. Refer to the SetPosition keyword manual page.</p>	<p>HomingDef[2]: Maximum speed.</p> <p>HomingDef[3]: Acceleration/Deceleration.</p> <p>HomingDef[4]: Emergency deceleration.</p> <p>HomingDef[5]: Velocity threshold to detect motor stuck.</p> <p>HomingDef[6]: Motor current (in mA) threshold to detect motor stuck.</p> <p>HomingDef[7]: Stuck time (in samples).</p> <p>HomingDef[8]: Position to set when the hard stop is detected.</p> <p>HomingDef[9]: Maximum step time, in [samples]</p>

Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
Move to hard stop (detected by high position error)	Move (using the provided motion parameters) till hard stop is detected using the provided parameters to detect high position error.	HomingDef[2]: Maximum speed.
	Motor stuck is detected when the position error (absolute value) is higher than the provided maximal position error threshold.	HomingDef[3]: Acceleration/Deceleration.  HomingDef[4]: Emergency deceleration.
	Once motor stuck is detected, the motor position is set to the given "position to set" value.	HomingDef[5]: Maximal position error threshold to detect motor stuck.
	The sign of the Maximum speed parameter defines the direction of the motion.	HomingDef[6]: Position to set when the hard stop is detected.
	Note that this step will not set the desired position in case the conditions for SetPosition are not met. Refer to the SetPosition keyword manual page.	HomingDef[7]: Maximum step time, in [samples]

Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
Jog to a change in the Home discrete input	Move (using the provided motion parameters) till a change in the Home discrete input is detected. Once detected, stop the motion.	HomingDef[2]: Maximum speed.
	The sign of the given Maximum speed parameter and the state of the Home discrete input defines the direction of the motion.	HomingDef[3]: Acceleration/Deceleration.  HomingDef[4]: Emergency deceleration.
	If the Home discrete input is "0" the sign of the Maximum speed parameter is the direction of the motion. If the Home is "1", the direction is inverted.	HomingDef[5]: Maximum step time, in [samples]
Move absolute PTP (point to point)	Move with the provided motion parameters to a given absolute target position.	HomingDef[2]: Maximum speed.  HomingDef[3]: Acceleration/Deceleration.  HomingDef[4]: Absolute target position.  HomingDef[5]: Maximum step time, in [samples]

Step type	Step description	Step parameters at HomingDef[2-10] (or [12-20], [22-30] ...)
Set software position limits	Provides the means to optionally set value to the reverse software position limit (RevPLim parameter) and similarly also for the forward limit.	<p>HomingDef[2]: Set value to RevPLim? 1 to set, 0 to avoid setting.</p> <p>HomingDef[3]: New value for RevPLim (ignored if HomingDef[2] is 0).</p> <p>HomingDef[4]: Set value to FwdPLim? 1 to set, 0 to avoid setting.</p> <p>HomingDef[5]: New value for FwdPLim (ignored if HomingDef[4] is 0).</p>

Most of the step types include built-in error detection mechanism. For example, timeout, or reaching an unexpected limit switch and so on. When such an error is detected during the homing process, the process is aborted and HomingStat is properly set to reflect the error type (see details above).

Note that upon entering the homing process, the controller motion parameters (Speed, Acceleration, deceleration and Emergency deceleration) are temporarily saved, and are restored when the homing process is completed. This is required since the homing process may change the values of these parameters.

Description	Value
<b>Mnemonic (keyword):</b>	HomingOn
<b>CAN code:</b>	340
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[HomingDef](#), [HomingStat](#), [MotionReason](#) and [ConFlt](#).

## HomingStat

Please refer to the [HomingOn](#) keyword page for detailed description of the Homing function at Agito's controllers.

Description	Value
<b>Mnemonic (keyword):</b>	HomingStat
<b>CAN code:</b>	342
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Homingon](#),

## la

la reports the value of phase “A” current in MilliAmperes. Phase “A” is according to the connection scheme as defined in the hardware reference guide.

Description	Value
<b>Mnemonic (keyword):</b>	la
<b>CAN code:</b>	9
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[laRef](#), [lb](#), [ld](#),



## laErr

laErr returns the value of phase “A” current error in milliAmperes. The current error is the difference between the phase current command (laRef) and the actual current (la).

Description	Value
<b>Mnemonic (keyword):</b>	laErr
<b>CAN code:</b>	20
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If laRef = 1000 and la = 990 then laErr = 100

### See Also:

[la](#), [laRef](#),

## laRef

laRef reports the current reference that is generated by the current loop to phase “A” of the motor. This value is in milliAmperes.

Description	Value
<b>Mnemonic (keyword):</b>	laRef
<b>CAN code:</b>	27
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[la](#), [laErr](#),

## Ib

Ib reports the value of phase “B” current in MilliAmperes. Phase “B” is according to the connection scheme as defined in the hardware reference guide.

Description	Value
<b>Mnemonic (keyword):</b>	Ib
<b>CAN code:</b>	10
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## IbErr

IbErr returns the value of phase “B” current error in milliAmperes. The current error is the difference between the phase current command (IbRef) and the actual current (Ib).

Description	Value
<b>Mnemonic (keyword):</b>	IbErr
<b>CAN code:</b>	21
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

If IbRef = 1000 and Ib = 990 then IbErr = 100

**See Also:**

N/A

## IbRef

IbRef returns the current reference of phase “B” as generated by the internal control loops.  
Phase “B” is as defined in the hardware user guide.

Description	Value
<b>Mnemonic (keyword):</b>	IbRef
<b>CAN code:</b>	28
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Id

In vector control mode (see ControlMode keyword), the current control loops are closed on Id and Iq. Iq closed loop takes CurrRef as its reference and Id closed loop takes zero as its reference.

Id is calculated from Ia and Ib, and is the non-effective current, meaning the current that does not create torque (in contrast to Iq which creates torque).

Id is given in [mA].

Note that Id is calculated even when the controller is not using Vector Control. The user can monitor and analyze Id and Iq to estimate the efficiency of the motor operation.

Typically Iq shall be equal to CurrRef and Id shall be equal to zero.

Note:

In older FW versions (earlier than version 1.3.0), Id and Iq were swapped. This is fixed, and is according to the above documentation, starting from firmware version 1.3.0.

Description	Value
<b>Mnemonic (keyword):</b>	Id
<b>CAN code:</b>	11
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Ia](#), [IdRef](#), [IdErr](#), [IqRef](#), [Iq](#), [IqErr](#),

## IdenResults

Refer to the description of the [CalcIden](#) keyword.

Description	Value
<b>Mnemonic (keyword):</b>	IdenResults
<b>CAN code:</b>	127
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 11
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[CalcIden](#),

## Identity

Identity[] is an array that includes information about the controller's identification, covering: hardware, firmware and manufacturing information, as follows:

Identity[1]: Product type  
Identity[2]: Serial number  
Identity[3]: Hardware version  
Identity[4]: Firmware version  
Identity[5]: FPGA version  
Identity[6]: Manufacturing date (DDMMYYYY)  
Identity[7]: Tested by (employee code)  
Identity[8]: Customer (numeric code)  
Identity[9]: Continuous current limit (mA)  
Identity[10]: Peak current limit (mA)  
Identity[11]: Minimum bus voltage (mV)  
Identity[12]: Maximum bus voltage (mV)  
Identity[13]: Current loop samples per second  
Identity[14]: Length of net data recording array  
Identity[15]: Maximum number of vectors to record  
Identity[16]: Number of axes  
Identity[17]: Boot program version  
Identity[18]: Maximum user program threads allowed  
Identity[19]: User program numeric stack depth

Description	Value
<b>Mnemonic (keyword):</b>	Identity
<b>CAN code:</b>	1
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 35
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

Identity [3]      Reports the HW version of the servo controller.

### See Also:

N/A



## IdErr

In vector control mode (see ControlMode keyword), the current control loops are closed on Id and Iq. Iq closed loop takes CurrRef as its reference and Id closed loop takes zero as its reference.

IdErr is (IdRef-Id), in [mA]

It is the error of the Id control loop in vector control mode.

Note that IdErr is calculated (just as IdRef and Id) even when the controller is not in vector control mode.

Description	Value
<b>Mnemonic (keyword):</b>	IdErr
<b>CAN code:</b>	22
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	FW1.3.0

**Example:**

**See Also:**

[Id](#), [IdRef](#),

## IdRef

In vector control mode (see ControlMode keyword), the current control loops are closed on Id and Iq. Iq closed loop takes CurrRef as its reference and Id closed loop takes zero as its reference.

IdRef is the reference for the Id control loop, in [mA]. It is actually always equal to CurrRef.

Note that IdRef is calculated even when the controller is not in vector control mode.

Description	Value
<b>Mnemonic (keyword):</b>	IdRef
<b>CAN code:</b>	29
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	FW1.3.0

**Example:**

**See Also:**

[Id](#), [IdErr](#),

## IndexPos

The IndexPos holds the last main encoder position at which an encoder index was detected.

TheIndexPos works only with incremental encoder.

Note that to properly detect an index signal and its position, the encoder velocity must be smaller than the controller sampling frequency, so that the index pulse width will be at least as wide as a single sampling.

Description	Value
<b>Mnemonic (keyword):</b>	IndexPos
<b>CAN code:</b>	47
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[StopOnIndex](#), [IndexStat](#), [AuxIndexStat](#) and [AuxIndexPos](#).

## IndexStat

The IndexStat holds the current status ("0" or "1") of the main encoder index input (for incremental encoder only).

Description	Value
<b>Mnemonic (keyword):</b>	IndexStat
<b>CAN code:</b>	45
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[StopOnIndex](#), [IndexPos](#), [AuxIndexStat](#) and [AuxIndexPos](#)

## IndirectArray

Description	Value
Mnemonic (keyword):	IndirectArray
CAN code:	436
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	No
Min value:	1
Max value:	1
Default value:	1
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

## IndirectDo

Description	Value
Mnemonic (keyword):	IndirectDo
CAN code:	437
Type:	Command
Access:	Read only
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	No
Min value:	0
Max value:	0
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

## IndirectIndex

Description	Value
<b>Mnemonic (keyword):</b>	IndirectIndex
<b>CAN code:</b>	434
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	1,000
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## IndirectValue

Description	Value
Mnemonic (keyword):	IndirectValue
CAN code:	435
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	No
Min value:	-2,147,483,648
Max value:	2,147,483,647
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:



## InjectCurrAmp

InjectCurrAmp sets the amplitude of the injection to the current reference in milliAmperes. If the injection is sinusoidal this will be the amplitude of the sine wave. If the required signal is a square wave or pulse this is the value of the signal when it is not zero.

Description	Value
<b>Mnemonic (keyword):</b>	InjectCurrAmp
<b>CAN code:</b>	114
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectPoint](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectedValue](#), [InjectType](#), [InjectTimeOn](#), [InjectCurrDC](#)

## InjectCurrDC

InjectCurrDC is the bias current in milliAmperes that is added to InjectCurrentAmp injected if InjectType = 5. After InjectTimeOn the current reference is equal to InjectCurrDC.

Description	Value
<b>Mnemonic (keyword):</b>	InjectCurrDC
<b>CAN code:</b>	126
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectPoint](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectedValue](#), [InjectType](#), [InjectTimeOn](#), [InjectCurrAmp](#),

## InjectedValue

InjectedValue is a read only parameter that can be used to monitor the value of the injected signal at any time.

Description	Value
<b>Mnemonic (keyword):</b>	InjectedValue
<b>CAN code:</b>	118
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectPoint](#), [InjectCurrAmp](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectType](#), [InjectTimeOn](#), [InjectCurrDC](#),

## InjectForceA

Description	Value
<b>Mnemonic (keyword):</b>	InjectForceA
<b>CAN code:</b>	590
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## InjectFreq

InjectFreq determines the frequency of the injection signal. To enable frequencies under 1Hz the entered value is internally divided by 100 to calculate the frequency in Hz.

Description	Value
<b>Mnemonic (keyword):</b>	InjectFreq
<b>CAN code:</b>	117
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	800,000
<b>Default value:</b>	2,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

InjectFreq = 20 will result in a frequency of  $20/100 = 0.2$  Hz

InjectFreq = 3000 will result in a frequency of 30

### See Also:

[InjectPoint](#), [InjectCurrAmp](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectType](#), [InjectTimeOn](#),  
[InjectCurrDC](#), [InjectedValue](#),

## InjectPoint

InjectPoint determines to which control loop the signal selected by InjectType will be injected.

The possible injection points are:

Value	Injection type
0	Inject current reference
1	Inject velocity reference
2	Inject position reference

Description	Value
<b>Mnemonic (keyword):</b>	InjectPoint
<b>CAN code:</b>	113
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectCurrAmp](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectedValue](#), [InjectType](#),  
[InjectTimeOn](#), [InjectCurrDC](#),

## InjectPosAmp

InjectPosAmp sets the amplitude of the injection to the position reference in user units. If the injection is sinusoidal this will be the amplitude of the sine wave. If the required signal is a square wave or pulse this is the value of the signal when it is not zero.

Description	Value
<b>Mnemonic (keyword):</b>	InjectPosAmp
<b>CAN code:</b>	116
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	100
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectPoint](#), [InjectCurrAmp](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectType](#), [InjectTimeOn](#), [InjectCurrDC](#), [InjectedValue](#),

## InjectTimeOn

InjectTimeOn is the duration in milliseconds of an injected pulse if InjectType = 5.

Description	Value
<b>Mnemonic (keyword):</b>	InjectTimeOn
<b>CAN code:</b>	125
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectType](#), [InjectPoint](#), [InjectCurrAmp](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectedValue](#),



## InjectType

InjectType determines which type of signal will be injected when using the injection feature. The possible injection types are:

Value	Injection type
0	No injection
1	Direct sine signal
2	Add sine signal
3	Direct square signal
4	Add square signal
5	Direct pulse

The required signal type is injected to the control loop that is selected by InjectPoint.

**Direct sine signal**

A sinusoidal signal is injected instead of any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, the position reference that is generated by the profiler is ignored and replaced by a sinusoidal reference.

**Add sine signal**

A sinusoidal signal is added any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, a sinusoidal signal is added to the position reference that is generated by the profiler.

**Direct square signal**

A square wave signal is injected instead of any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, the position reference that is generated by the profiler is ignored and replaced by a square wave reference. This can be used as a simulated step response.

**Add square signal**

A square wave signal is added any other reference that is supposed to be used in the injection point. For example, if the injected signal is injected to the position loop, a square wave signal is added to the position reference that is generated by the profiler.

**Direct pulse**

A single pulse is injected instead of any other reference that is supposed to be used in the injection point.

Description	Value
<b>Mnemonic (keyword):</b>	InjectType
<b>CAN code:</b>	112
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes

<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	6
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectPoint](#), [InjectCurrAmp](#), [InjectVelAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectType](#), [InjectTimeOn](#),  
[InjectCurrDC](#),

## InjectVelAmp

InjectVelAmp sets the amplitude of the injection to the velocity reference in user units per second. If the injection is sinusoidal this will be the amplitude of the sine wave. If the required signal is a square wave or pulse this is the value of the signal when it is not zero.

Description	Value
<b>Mnemonic (keyword):</b>	InjectVelAmp
<b>CAN code:</b>	115
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	10,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InjectPoint](#), [InjectCurrAmp](#), [InjectPosAmp](#), [InjectFreq](#), [InjectType](#), [InjectTimeOn](#), [InjectCurrDC](#), [InjectedValue](#),

## InTargetStat

InTargetStat indicates the progress of the motion until the target is reached. If the position error in user units is less than InTargetTol for at least InTargetTime milliseconds then InTargetStat receives the value that indicates that the target was reached.

The values of InTargetStat:

Value	Status
0	Motor off
1	Motor on
2	In motion
3	Waiting for InTargetTime to elapse
4	Target reached

Description	Value
Mnemonic (keyword):	InTargetStat
CAN code:	268
Type:	Parameter
Access:	Read only
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	Yes
Min value:	0
Max value:	4
Default value:	0
User units:	No user units
Implementation status:	Implemented

**Example:**

**See Also:**

[InTargetTime](#), [InTargetTol](#),

## InTargetTime

InTargetTime is the time in milliseconds that the motor should be within tolerance from the target in order to determine that the target was reached. If the position error in user units is less than InTargetTol for at least InTargetTime milliseconds then InTargetStat receives the value that indicates that the target was reached.

Description	Value
<b>Mnemonic (keyword):</b>	InTargetTime
<b>CAN code:</b>	266
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000
<b>Default value:</b>	3
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InTargetStat](#), [InTargetTol](#),

## InTargetTol

InTargetTol is the “In target” tolerance. If the position error in user units is less than InTargetTol for at least InTargetTime milliseconds then InTargetStat receives the value that indicates that the target was reached.

Description	Value
<b>Mnemonic (keyword):</b>	InTargetTol
<b>CAN code:</b>	265
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	10
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[InTargetTime](#), [InTargetStat](#),

## InTargetVelTh

Description	Value
<b>Mnemonic (keyword):</b>	InTargetVelTh
<b>CAN code:</b>	292
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	1,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Iq

In vector control mode (see ControlMode keyword), the current control loops are closed on Id and Iq. Iq closed loop takes CurrRef as its reference and Id closed loop takes zero as its reference.

Iq is calculated from Ia and Ib, and is the effective current, meaning the current that creates torque (in contrast to Id which does not create torque).

Iq is given in [mA].

Note that Iq is calculated even when the controller is not using Vector Control. The user can monitor and analyze Id and Iq to estimate the efficiency of the motor operation.

Typically Iq shall be equal to CurrRef and Id shall be equal to zero.

Note:

In older FW versions (earlier than version 1.3.0), Id and Iq were swapped. This is fixed, and is according to the above documentation, starting from firmware version 1.3.0.

Description	Value
<b>Mnemonic (keyword):</b>	Iq
<b>CAN code:</b>	12
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	FW1.3.0

**Example:**

**See Also:**

[IdRef](#), [Id](#), [IdErr](#), [IqRef](#), [IqErr](#), [ControlMode](#),



## IqErr

In vector control mode (see ControlMode keyword), the current control loops are closed on Id and Iq. Iq closed loop takes CurrRef as its reference and Id closed loop takes zero as its reference.

IqErr is (IqRef-Iq), in [mA]

It is the error of the Iq control loop in vector control mode.

Note that IqErr is calculated (just as IqRef and Iq) even when the controller is not in vector control mode.

Description	Value
<b>Mnemonic (keyword):</b>	IqErr
<b>CAN code:</b>	23
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	FW1.3.0

**Example:**

**See Also:**

[IdRef](#), [Id](#), [IdErr](#), [IqRef](#), [Iq](#), [ControlMode](#),

## IqRef

In vector control mode (see ControlMode keyword), the current control loops are closed on Id and Iq. Iq closed loop takes CurrRef as its reference and Id closed loop takes zero as its reference.

IqRef is the reference for the Iq control loop, in [mA]. It is actually always equal to zero.

Note that IqRef is calculated even when the controller is not in vector control mode.

Description	Value
<b>Mnemonic (keyword):</b>	IqRef
<b>CAN code:</b>	30
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	1.3.0

**Example:**

**See Also:**

[IdRef](#), [Id](#), [IdErr](#), [Iq](#), [IqErr](#), [ControlMode](#),

## Jerk

---

Jerk defines the time it takes to build the desired acceleration (or deceleration) in all profiler-based motions.

The time to build the acceleration is calculated by:

$$\text{Jerk Time} = \text{SAMPLE\_TIME} * 2\text{Jerk}$$

SAMPLE\_TIME is the sampling time of the controller, typically  $1/16384=61\mu\text{s}$  with Agito controllers.

So, for example, if JERK = 7, the jerk time is 7.808ms.

Note that the Jerk is the only motion profile parameter that cannot be modified during the motion.

The Jerk Time is used whenever acceleration starts or ends, and similarly for the deceleration.

Clearly, with higher values of Jerk, the motion time is longer. However, it is smoother and reduced overshoots are expected. The jerk shall be tuned for optimal performance per each specific application.

### Notes:

Smoothing (Jerk not 0) can be used together with endless motion (ModRev not 0), but the user must take care to properly set Jerk and ModRev, according to the following guidelines:

The time (in number of samples) for a full revolution of the modulus (moving ModRev distance), at the maximal speed, should be longer than the time associated with the Jerk value (2Jerk is the number of samples).

This can be easily solved by properly setting ModRev (large enough).

For example, if the maximal speed is 10,000,000 [counts/sec] and Jerk = 9 (i.e.  $29 = 512$  samples), and the controller sampling rate is 16,384 [samples/sec], then ModRev must be higher than  $10,000,000 / 16,384 * 512 = 312,500$  [counts].

The multiplication of the number of samples associated with the Jerk (it is 2Jerk [samples]) and ModRev (in [counts]) must be smaller than (231-1).

This can be easily solved by properly setting ModRev (small enough).

Continuing the above example, if Jerk = 9, then the ModRev must be smaller than:  $(231-1) / 512 = 4,194,304$  [counts]

Please consult Agito for the proper way to set ModRev and Jerk, for a given application that requires smoothing and endless motion together, if you, from some reason, can't set ModRev as high as needed or as low as needed, according to the above guidelines.

The Jerk (smoothing) value appears at the PC Suite at almost all motion windows. However, please note that the Jerk value is not written to the controller when a motion button is pressed (unlike other parameters like acceleration and speed). This is because the Jerk cannot be modified while motor is on. So, if a new value for smoothing (jerk) is needed, the user should

change the smoothing value, and then disable the motor, use “Apply Changes” to write the new Jerk value (it will be now written, as the motor is disabled) and only then use the motion buttons (the motion will now use the new smoothing/Jerk value).

Description	Value
<b>Mnemonic (keyword):</b>	Jerk
<b>CAN code:</b>	139
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	9
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Accel](#), [Decel](#), [Speed](#), [RelTrgt](#), [AbsTrgt](#) and [ModRev](#).

## Jump

Jump is a user program low level language keyword. The syntax related to “Compare” can only generated automatically by the PC suite during compilation. The information in the command includes a pointer for the jump that depends on the structure of the program file.

Description	Value
<b>Mnemonic (keyword):</b>	Jump
<b>CAN code:</b>	196
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 9
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	131,072
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## LampFullScale

Description	Value
<b>Mnemonic (keyword):</b>	LampFullScale
<b>CAN code:</b>	229
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## LAmPVBus

Description	Value
<b>Mnemonic (keyword):</b>	LAmPVBus
<b>CAN code:</b>	253
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## LimitsStat

The LimitsStat parameter reflects the current status of the RLS (Reverse Limit Switch) and the FLS (Forward Limit Switch) inputs.

LimitsStat (bit 0) reflects the status of RLS ("0" not activated, "1" activated).

LimitsStat (bit 1) reflects the status of FLS ("0" not activated, "1" activated).

Bit 0 is the rightmost bit of the parameter.

Description	Value
<b>Mnemonic (keyword):</b>	LimitsStat
<b>CAN code:</b>	49
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[DInPot](#), [DInLog](#), [DInMode](#), [MotionStat](#) and [MotionReason](#).



## Lm

Description	Value
<b>Mnemonic (keyword):</b>	Lm
<b>CAN code:</b>	374
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	100,000
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Load

Load is used to retrieve all the parameters from the non-volatile (flash) memory. If the user changes the values of some parameters and then wants to return to a previously saved set of parameters, Load will bring the entire set from the flash.

Load is also performed internally upon reset so all the parameters that were saved to flash will have their saved values after reset.

Description	Value
<b>Mnemonic (keyword):</b>	Load
<b>CAN code:</b>	233
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

Set Speed = 100000

Save

Query Speed to receive 100000.

Now change Speed to 150000

Query to make sure the change took effect.

Now Load.

If you query Speed it will return the saved value of 100000.

### See Also:

[Save](#).

## LockCntr

Lock counter reports the number of times a change in the input designated as lock source was detected since the latest enabling of lock. The value is reset by LockEn = 0, LockEn = 1. It can also be reset by assigning 0.

Description	Value
<b>Mnemonic (keyword):</b>	LockCntr
<b>CAN code:</b>	172
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[LockEn](#), [LockVal](#), [LockCntr](#), [LockSrc](#).

## LockEn

LockEn = 1 will enable the position lock. When position lock is enabled a change in the designated input (rise for CW motion, fall for CCW) will lock the main encoder position. Lock and Event are mutually exclusive functions, so when lock is enabled event will be automatically disabled.

The value of the position during the input change is saved in "LockVal". The number of times a position was locked is saved in "LockCnt".

For incremental encoder, the position is locked by hardware, so it is very accurate. For absolute encoder the position that is read in the next sampling after the lock input changed is saved in "LockVal".

LockVal and LockCnt are updated only once every sample time. If more than one valid change in the input happens during a single sample time only the last change will be recorded.

Description	Value
<b>Mnemonic (keyword):</b>	LockEn
<b>CAN code:</b>	170
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[LockVal](#), [LockCnt](#), [LockSrc](#).

## LockSrc

LockSrc is the number of input used as source for the lock feature. Changes on this designated input will be reflected in LockVal and LockCntr. The same input that is designated as lock source can also have a different role assigned to it in DInMode.

Description	Value
<b>Mnemonic (keyword):</b>	LockSrc
<b>CAN code:</b>	171
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-32
<b>Max value:</b>	32
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

Input 3 can be left limit switch, and also be assigned the role of lock source. This will enable performing a homing procedure that will find the position of the limit switch.

### See Also:

[LockEn](#), [LockVal](#), [LockCntr](#), [DInMode](#).

## LockVal

LockVal holds the value of the latest position where the lock input changed. This value is saved by hardware for an incremental encoder. For absolute encoder the value of the position during the next sample is saved.

Description	Value
<b>Mnemonic (keyword):</b>	LockVal
<b>CAN code:</b>	173
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[LockEn](#), [LockSrc](#), [LockCntr](#).

## LockValTable

Description	Value
<b>Mnemonic (keyword):</b>	LockValTable
<b>CAN code:</b>	315
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 100
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MapEncoder

Refer to the detailed description of error mapping under the MapType keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	MapEncoder
<b>CAN code:</b>	322
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 3
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	12
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## MapLength

Refer to the detailed description of error mapping under the [MapType](#) keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	MapLength
<b>CAN code:</b>	324
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 3
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	3,000
<b>Default value:</b>	10
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MapType](#),

## MapPosGap

Refer to the detailed description of error mapping under the [MapType](#) keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	MapPosGap
<b>CAN code:</b>	325
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 3
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	8,000,000
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MapType](#),

## MapStartIndex

Refer to the detailed description of error mapping under the [MapType](#) keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	MapStartIndex
<b>CAN code:</b>	321
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	3,000
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MapType](#),

## MapStartPos

Refer to the detailed description of error mapping under the [MapType](#) keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	MapStartPos
<b>CAN code:</b>	323
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 3
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MapType](#),

## MapTable

Refer to the detailed description of error mapping under the MapType keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	MapTable
<b>CAN code:</b>	326
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 3000
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MapType](#),

## MapType

The feedback encoder is assumed to provide the accurate position of the motor (or the load, depending on the machine structure).

However, in some application, the encoder reading does not accurately measure the motor or (more relevant) the load position.

Why? Many reasons can be considered. For example:

1. The encoder is attached to the motor, while the mechanical transmission between the motor and the lost (whose position is of interest) is not constant.
2. The mechanical structure of the system is not linear (decentralized wheel transmission).
3. The system is an XY system, and the XY are not exactly 90 degrees from each other.

For such cases, Agito's controllers support 1D (1 dimensional) and 2D (2 dimensional) encoder error corrections mechanism.

The user can define a map (table) of the position reading errors, and the controller will automatically (each sample time) correct the encoder reading so that the final position reading (at the Pos keyword) will have a corrected value, reflecting, as much as possible, the actual position of the motor/load.

Note that since the position reading correction is done at each sample time, and is done on the feedback reading, it does not only correct the final motion position, but also the velocity along the motion.

The error mapping is a relatively complex feature. It is strongly recommended to use the PC Suite (Feedback/Error Mapping window) to access and to use this functionality.

Two operational modes are supported:

- 1D Error mapping:

The main encoder reading is corrected as a function of a single encoder input (typically the main encoder itself) and a list of corresponding errors.

- 2D error mapping:

The main encoder reading is corrected as a function of two encoder inputs and a 2D table of corresponding errors.

The relevant parameters are:

- MapType:

If MapType == 0 Error correction is disabled.

If MapType == 1 1D error mapping is activated.

If MapType == 2 2D error mapping is activated.

Use this parameter to enable/disable the error correction/mapping and to define its mode (1D or 2D).

- MapEncoder[]:

MapEncoder[] defines which encoder feedback is used as an entry value into the error correction/mapping table.

MapEncoder[] is an array with two entries. MapEncoder[1] is used for 1D error mapping and also as the first encoder for 2D error mapping. MapEncoder[2] is only used at 2D error mapping, used as the second encoder definition.

If MapEncoder[1/2] == 1            The main encoder of the **A** axis is used as the entry to the table.

If MapEncoder[1/2] == 2            The auxiliary encoder of the **A** axis is used as the entry to the table.

For multi-axes controllers only:

If MapEncoder[1/2] == 3            The main encoder of the **B** axis is used as the entry to the table.

If MapEncoder[1/2] == 4            The auxiliary encoder of the **B** axis is used as the entry to the table.

And so on ...

■ MapTable[]:

MapTable[] is a large array that holds the table of errors to be used for the error correction.

For 1D error correction, the user shall place a list of values (error values) at any location within the MapTable[]. The MapStartIndex shall point to the index at MapTable[] where the first element in this list is stored.

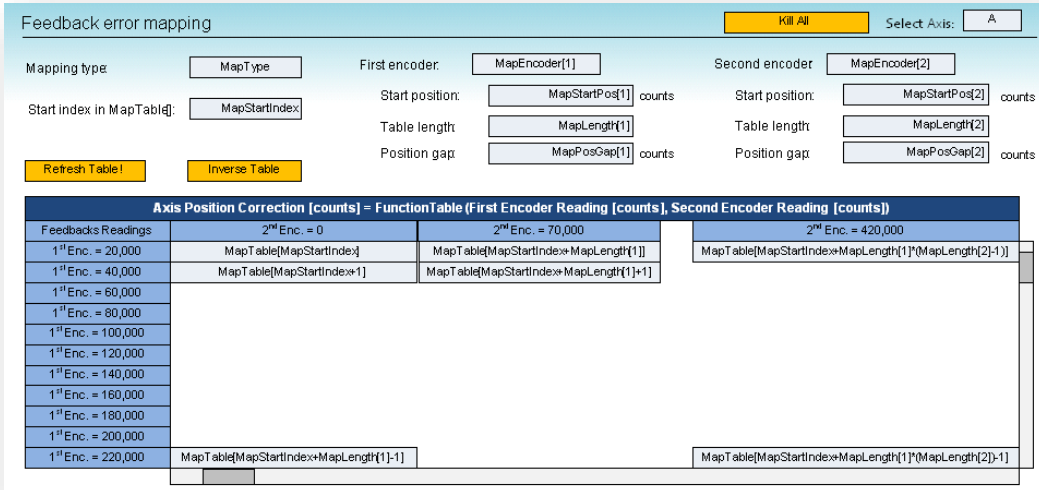
Note that MapTable[] starts at MapTable[1] (like all arrays at Agito's controllers.

Each error in the list corresponds to a given input value at the (selected) encoder input.

It is assumed that the first point (MapTable[MapStartIndex[1]]) corresponds to MapStartPos[1] and the following points correspond to equally spaced input positions, using MapPosGap[1]. The number of table entries is defined by MapLength[1].

For 2D error correction, the MapTable is built from a list of errors as defined above, per each value of the second encoder input (which starts at MapStartPos[2] and increments by MapPosGap[2], for a MapLength[2] number of points). The 2D table is organized within MapTable[] (which is a 1D array) as a series of lists. First is the list for the first value of the second encoder. Then comes the list of the second value and so on.

Refer to:



Feedback error mapping

Mapping type:  First encoder:  Second encoder:  Select Axis:

Start index in MapTable:  Start position:  counts Start position:  counts

Table length:  Table length:  counts

Position gap:  counts Position gap:  counts

Axis Position Correction [counts] = FunctionTable (First Encoder Reading [counts], Second Encoder Reading [counts])			
Feedbacks Readings	2 <sup>nd</sup> Enc. = 0	2 <sup>nd</sup> Enc. = 70,000	2 <sup>nd</sup> Enc. = 420,000
1 <sup>st</sup> Enc. = 20,000	MapTable[MapStartIndex]	MapTable[MapStartIndex+MapLength[1]]	MapTable[MapStartIndex+MapLength[1]*MapLength[2]-1]
1 <sup>st</sup> Enc. = 40,000	MapTable[MapStartIndex+1]	MapTable[MapStartIndex+MapLength[1]+1]	
1 <sup>st</sup> Enc. = 60,000			
1 <sup>st</sup> Enc. = 80,000			
1 <sup>st</sup> Enc. = 100,000			
1 <sup>st</sup> Enc. = 120,000			
1 <sup>st</sup> Enc. = 140,000			
1 <sup>st</sup> Enc. = 160,000			
1 <sup>st</sup> Enc. = 180,000			
1 <sup>st</sup> Enc. = 200,000			
1 <sup>st</sup> Enc. = 220,000	MapTable[MapStartIndex+MapLength[1]-1]		MapTable[MapStartIndex+MapLength[1]*MapLength[2]-1]

- MapStartIndex:

The index of the first element at MapTable[] that holds the beginning of the errors table. Typically use a value of 1, unless you would like to store multiple corrections tables at MapTable[] (this is possible and limited only by the size of the MapTable[] array).

- MapStartPos[]:

MapStartPos[1] defines the input encoder reading that corresponds to the beginning of the table for 1D error correction and the input value of the 1<sup>st</sup> encoder at the first point of the table in 2D error correction.

MapStartPos[2] is used only for 2D error mapping and defines input value of the 2<sup>nd</sup> encoder at the first point of the table in 2D error correction.

- MapLength[]:

MapLength[1] defines the number of entries at the errors table, for 1D error mapping.

MapLength[1], for 2D error mapping, defines the number of entries that relates to the 1<sup>st</sup> encoder input (the number of "rows" at the 2D error mapping table).

MapLength[2], is used only for 2D error mapping, and defines the number of entries that relates to the 2<sup>nd</sup> encoder input (the number of "columns" at the 2D error mapping table).

- MapPosGap[]:

MapPosGap[1], for 1D error mapping, defines the input encoder gap (in [counts]) between two consecutive entries to the table.

For 2D error mapping, MapPosGap[1] defines the gaps of the 1<sup>st</sup> encoder, while MapPosGap[2] (used only in 2D error mapping) defines the gap for the 2<sup>nd</sup> encoder.

How the corrected position reading value is calculated?



For 1D error mapping:

- The relevant encoder input is read (it can be the main encoder or the auxiliary encoder, but typically it is the main encoder of this axis).
- If the encoder input value is smaller than MapStartPos[1], the first error value is used (MapTable[MapStartIndex]).
- Else if the encoder input value is higher than MapStartPos[1]+(MapLength[1]-1)\*MapPosGap[1] the last error is used (MapTable[MapStartIndex + MapLength[1] - 1]).
- Otherwise, the related pointer into the table is calculated and a linear interpolation is used between the relevant two table entries, to calculate the resulted error value.
- The resulted error value is **added** to the main encoder reading, resulting with the feedback position at the Pos parameter. Note that the error is added to the encoder reading (this affects the way the user shall measure and calculate the table of errors).

For 2D error mapping:

- The process is very similar, but:
- The process is first applied on the 1<sup>st</sup> encoder.
- The process is then applied on the 2<sup>nd</sup> encoder.
- Now that we have the rectangle within the error is located, a process of 3 linear interpolations is executed in order to find the resulted 2D error result.
- This error is now used as described above to correct the main encoder reading.

Description	Value
<b>Mnemonic (keyword):</b>	MapType
<b>CAN code:</b>	320
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MapTable](#), [MapEncoder](#), [MapStartIndex](#), [MapStartPos](#), [MapLength](#), [MapPosGap](#), [Pos](#), [AuxPos](#).

## MasterFact

MasterFact is the gearing ratio. In gearing mode the master input is multiplied by MasterFact to generate the gearing master reference.

To enable the use of fractions MasterFact is multiplied by 65536. For a gearing ratio of 1 MasterFact = 65536. For a gearing ratio of 0.5 MasterFact = 32768. For a gearing ratio of 3 MasterFact =  $65536 * 3 = 196608$ .

Description	Value
<b>Mnemonic (keyword):</b>	MasterFact
<b>CAN code:</b>	120
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-16,777,215
<b>Max value:</b>	16,777,215
<b>Default value:</b>	65,536
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MasterPos](#), [MotionMode](#),

## MasterFilt

MasterFilt is used in Direct Gearing motion mode. In this mode, the master position (MasterPos) is calculated using the auxiliary encoder input (AuxPos), multiplied by the master factor (MasterFact).

This master position is now filtered using a first order digital filter and is used as the position reference for the control loop.

The filter is required (application dependent) to avoid large “jumps” at the position reference, especially in cases of large master factor.

MasterFilt defines the filter frequency.

The digital filter is:

$$\text{PosRef}_k = (\text{MasterPos}_k * \text{MasterFilt} + \text{PosRef}_{k-1} * (64 - \text{MasterFilt})) / 64;$$

Description	Value
<b>Mnemonic (keyword):</b>	MasterFilt
<b>CAN code:</b>	161
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	64
<b>Default value:</b>	3
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MasterPos](#), [MasterFact](#), [AuxPos](#).

## MasterPos

MasterPos is a read only parameter that returns the position of the master for gearing multiplied by the gearing ratio.

Description	Value
<b>Mnemonic (keyword):</b>	MasterPos
<b>CAN code:</b>	44
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MasterFact](#), [MotionMode](#).

## Math

Math is a user program low level language keyword. The syntax related to “Math” is usually generated automatically by the PC suite during compilation.

Math will perform the mathematical operation requested on the top parameter or parameters of the numeric stack. The number of parameters depends on the operation. For example: addition requires two parameters while negate is an operation that is performed on one parameter.

The index value determines which operation will be performed. The operands should be pushed to the stack before Math is called.

The result is pushed to the numeric stack. If this function is called from communication the value is also sent through communication.

Note that values in the stack and over communication are integers. Results in fractions will be rounded.

Internally all the relevant mathematical operation are performed on long long type variables.

The definitions are compatible with C language.

Pop1 is the first value that is popped from the stack (“top” value)

Pop2 is the second value that is popped from the stack

The values of math:

Value	Operation Type	Number of parameters
1	Add: Result = Pop1 + Pop2	2
2	Subtract : Result = Pop2-Pop1	2
3	Multiply: Result = Pop1 * Pop2	2
4	Divide: Result = Pop2 / Pop1	2
5	Negate: Result = -Pop1	1
6	Invert: Result = 1/Pop1 *	1
7	Modulo: Result = Pop2%Pop1	2
8	Power: Result = Pop2^Pop1 (^ used as power operator)	2
9	Square root: Result = Square root(Pop1)	1
10	Sine: Result = sin(Pop1)*	1
11	Cosine: Result = cos(Pop1)*	1
12	Tangent: Result = tan(Pop1)*	1
13	Cotangent: Result = tan <sup>-1</sup> ( Pop1)*	1
14	Inverse sine: Result = arcsin(Pop1)*	1
15	Inverse cosine: Result = arcos(Pop1)*	1
16	Inverse tangent: Result = arctan(Pop1)*	1
17	Bitwise not: Result = ~ Pop1	1
18	Bitwise and: Result = Pop1 & Pop2	2
19	Bitwise or: Result = Pop1   Pop2	2
20	Bitwise xor: Result = Pop1 ^ Pop2 (^ used as xor as in C )	2
21	Shift left: Result = Pop1 << Pop2	2
22	Shift right: Result = Pop1 >> Pop2	2
23	Absolute: Result = abs(Pop1)	1
24	Logarithm: Result = log <sub>pop2</sub> Pop1 *	2
25	Base 10 logarithm: Result = log <sub>10</sub> Pop1 *	1

26	Exponential: Result = $e^{\text{Pop1} *}$	1
27	Inverse tangent 2 *	
28	Logical and: Result = Pop1 && Pop2 (true = 1, false = 0)	2
29	Logical or: Result = Pop1    Pop2 (true = 1, false = 0)	2
30	Logical not: Result = ! Pop1 (true = 1, false = 0)	1
31	Pointed by complex CAN code Result = Value of parameter pointed by (Complex CAN code) Pop1 **  Note that the value returned by this math operation is the internal value of the pointed parameter, without internal scaling (if any) and user units conversion (if defined).	1

(\*) To be implemented and updated

(\*\*) Supported from FW 1.4.0 and higher.

Description	Value
<b>Mnemonic (keyword):</b>	Math
<b>CAN code:</b>	206
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 30
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MaxAcc

MaxAcc is the maximum acceleration allowed by Agito PC suite motion window. In this window the user can enter the desired acceleration as a percent of this value. It is saved in the controller but not used by it, so it is possible for the user to set a higher acceleration while ignoring this value.

Description	Value
<b>Mnemonic (keyword):</b>	MaxAcc
<b>CAN code:</b>	81
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MaxForceErr

Description	Value
<b>Mnemonic (keyword):</b>	MaxForceErr
<b>CAN code:</b>	585
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	327,680
<b>Default value:</b>	2,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## MaxMotorCurr

MaxMotorCurr is the maximal motor current allowed, in [mA].

The equivalent motor current is calculated from the measured phase A and phase B currents.

In case the axis is enabled, and the measured/calculated motor current (absolute value) is larger than the MaxMotorCurr parameter for 4 consecutive readings, the axis is disabled with a proper error code (at ConFlt).

The error is also logged at the ErrLog.

Description	Value
<b>Mnemonic (keyword):</b>	MaxMotorCurr
<b>CAN code:</b>	99
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Ia](#), [Ib](#), [MotorCurr](#), [ConFlt](#) and [ErrLog](#).

## MaxPhaseCurr

MaxPhaseCurr is the maximum allowed current per each individual motor phase in milliAmperes. If the current of any phase exceeds this value for more than 0.25 mSec, the motor is disabled. This protection is needed in cases where the total current through the motor is below the rating of the maximum current but the way the current is distributed between the phases causes one phase to heat.

Description	Value
<b>Mnemonic (keyword):</b>	MaxPhaseCurr
<b>CAN code:</b>	98
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MaxPosErr

MaxPosErr determines the maximum allowed position error. The position error is the difference between the position reference (PosRef) and the actual position of the motor (Pos). If this difference Exceeds MaxPosErr the motor is disabled. This parameter is used as a protection in case the control is not functioning.

MaxPosErr is in user units.

Description	Value
<b>Mnemonic (keyword):</b>	MaxPosErr
<b>CAN code:</b>	84
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	327,680
<b>Default value:</b>	20
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosErr](#), [Pos](#).

## MaxPWM

MaxPWM can be used to limit the maximum voltage output to the motor. The voltage output is implemented using a PWM signal. The width of the “ON” signal is determined by a value that is written to a counter. A high value to the counter will result in a high voltage to the motor phases. The value that is written will not exceed the value of MaxPWM. The default value of this parameter is also its maximum value, and it represents 100% of the rated voltage applied to the motor. To limit the voltage to a lower value lower MaxPWM to the percentage needed.

Description	Value
<b>Mnemonic (keyword):</b>	MaxPWM
<b>CAN code:</b>	91
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

### Example:

If the default value of MaxPWM is 4500, the rated voltage of the drive is 40V and the required maximum voltage is 20 V, assign MaxPWM = 2250.

### See Also:

## MaxPwrTemp

MaxPwrTemp is the maximum allowed temperature in Celsius for the power generating components of the product.

Description	Value
Mnemonic (keyword):	MaxPwrTemp
CAN code:	90
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	Yes
Axis related:	Yes
Min value:	20
Max value:	95
Default value:	65
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

## MaxVBus

MaxVBus is the maximum allowed bus voltage in millivolts. This parameter can be used to limit the bus voltage to a value that is lower than the rating of the product. If the actual bus voltage exceeds this value for a time that is greater than the time designated in MaxVBusTime the motor is disabled (MotorOn = 0)

Description	Value
<b>Mnemonic (keyword):</b>	MaxVBus
<b>CAN code:</b>	92
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MaxVbusTime](#), [MaxVbusAbs](#).

## MaxVBusAbs

MaxVBusAbs is the absolute maximum allowed bus voltage in milliamperes. If the bus voltage exceeds this value the motor is immediately disabled.

Description	Value
<b>Mnemonic (keyword):</b>	MaxVBusAbs
<b>CAN code:</b>	94
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MaxVbus](#), [MaxVbusTime](#).

## MaxVBusTime

MaxVBusTime determines the time in milliseconds that the bus voltage is allowed to exceed the value set in MaxVBus.

Description	Value
<b>Mnemonic (keyword):</b>	MaxVBusTime
<b>CAN code:</b>	93
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	3,051
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MaxVbus](#), [MaxVBusAbs](#).



## MaxVel

MaxVel determines the maximum allowed velocity for the controlled motor. The velocity reference from the velocity loop is limited by MaxVel. If the actual reading from the feedback exceeds MaxVel by more than 25% the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	MaxVel
<b>CAN code:</b>	80
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MaxVelErr

MaxVelErr is the maximum allowed velocity error. If the velocity error exceeds MaxVelErr the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	MaxVelErr
<b>CAN code:</b>	85
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	32,768
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VelErr](#),

## MinVBus

MinVBus defines the minimum bus voltage required for the motor to be enabled. If the bus voltage drops below this point the motor will be disabled and a fault is recorded.

MinVbus is in A/D output units for user 1.

In other products MinVbus is in milliVolts.

The minimum allowed value is the absolute minimum allowed for the product and also the default value. The user can raise the minimum value according to other considerations.

Description	Value
<b>Mnemonic (keyword):</b>	MinVBus
<b>CAN code:</b>	89
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ModRev

ModRev is used for the modulo mode of the controller. In the modulo mode the main position (Pos) is between 0 and ModRev. When the actual position exceeds ModRev the value of Pos is calculated as part of ModRev. This allows a rotary axis to move in the same direction indefinitely without exceeding any numerical limits.

ModRev = 0 turns off the Modulo mode.

Notes: Do not use Modulo together with input shaping.

If you manually set the position using SetPosition to a value that exceeds the range of ModRev unexpected behavior may occur.

Endless motion (ModRev not 0) can be used together with smoothing (Jerk not 0), but the user must take care to properly set ModRev and Jerk, according to the following guidelines:

The time (in number of samples) for a full revolution of the modulus (moving ModRev distance), at the maximal speed, should be longer than the time associated with the Jerk value (2Jerk is the number of samples).

This can be easily solved by properly setting ModRev (large enough).

For example, if the maximal speed is 10,000,000 [counts/sec] and Jerk = 9 (i.e.  $29 = 512$  samples), and the controller sampling rate is 16,384 [samples/sec], then ModRev must be higher than  $10,000,000 / 16,384 * 512 = 312,500$  [counts].

The multiplication of the number of samples associated with the Jerk (it is 2Jerk [samples]) and ModRev (in [counts]) must be smaller than  $(2^{31}-1)$ .

This can be easily solved by properly setting ModRev (small enough).

Continuing the above example, if Jerk = 9, then the ModRev must be smaller than:

$(2^{31}-1) / 512 = 4,194,304$  [counts]

Please consult Agito for the proper way to set ModRev and Jerk, for a given application that requires smoothing and endless motion together, if you, from some reason, can't set ModRev as high as needed or as low as needed, according to the above guidelines.

Description	Value
<b>Mnemonic (keyword):</b>	ModRev
<b>CAN code:</b>	70
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

---

**Example:**

**See Also:**

[Jerk](#),

## MotionMode

MotionMode determined the type of motion that will be performed after the next Begin command. MotionMode cannot be changed until the current motion ends.

The values of MotionMode and the related motion type are:

MotionMode

Resulting Motion

0

Jog – The motor will reach the velocity set in Speed and continue in constant speed until a Stop command is received. The acceleration and deceleration used for the motion are as defined by Accel and Decel. The motion may also be smoothed according to the value of Jerk.

1

Point to Point – Move to the position defined by RelTrgt of AbsTrgt (if RelTrgt = 0). The other parameters of the motion are as defined by Accel, Speed, Decel and Jerk.

2

Point to point repetitive – A repetitive motion from the current position to the the position defined by RelTrgt of AbsTrgt (if RelTrgt = 0) and back. The other parameters of the motion are as defined by Accel, Speed, Decel and Jerk.

The motor will wait at each end of the motion for the duration determined by RptWait. The motion is stopped using StopRep.

3

Pulse Direction indirect mode – The position command to this motion is determined according to the pulse – direction input. However, the number of pulses received is not immediately used as the reference to the position loop.

The number of pulses that were received is added to the current position target. The profiler builds a motion trajectory to the new target using the values of all the motion parameters: Accel, Decel, Speed, Jerk.

This means that even if a large number of input pulses is entered at once, the resulting motion will not look like a step response, but will be smoother. Note that the response to fast input will be smoother, but also slower.

4

Pulse Direction direct mode – The position reference to the position loop is received directly from the pulse direction input.

Description	Value
<b>Mnemonic (keyword):</b>	MotionMode
<b>CAN code:</b>	141
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-1
<b>Max value:</b>	16
<b>Default value:</b>	-1

---

<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

---

**Example:**

**See Also:**

[Accel](#), [Decel](#), [Speed](#), [Jerk](#), [AbsTrgt](#), [RelTrgt](#), [Begin](#), [RptWait](#), [StopRep](#).

## MotionReason

MotionReason returns a value that represents the reason that caused the end of the latest motion. The Begin command resets MotionReason to 0.

The table below lists the possible values of MotionReason and their meanings

Value	Meaning
0	Current motion still not ended or Motion Ended normally
1	Motion ended due to Stop command
2	Motion ended due to Abort command
3	Motion ended due to StopRep command
4	Motion ended due to reverse limit switch detection
5	Motion ended due to forward limit switch detection
6	Motion ended due to reverse software limit
7	Motion ended due to forward software limit
8	Motion ended due to motor disable
9	Motion ended due to a StopECAM command (for ECAM motions only)
10	Motion ended due to a StopFIFO command (for FIFO motions only)
11	Motion ended due to detected index (for Jogging only)

Description	Value
<b>Mnemonic (keyword):</b>	MotionReason
<b>CAN code:</b>	43
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If the motion was ended by an abort command, but during deceleration the forward software limit was exceeded, and then the limit switch was encountered, MotionReason will have a value of 2, indicating the original reason to stop and ignoring any following events that could have stopped the motion.

### See Also:



## MotionSamples

MotionSamples[] is an array parameter that reports the time duration of the last (recently completed) motion.

The time is reported in number of controller samples (typically for Agito's controllers: 1/16384=61(s per sample).

Three times are reported, as follows:

- MotionSamples[1]: The time of the motion profile itself.  
Independent of the actual motion of the motor/axis.
- MotionSamples[2]: The time of the motion profile itself plus the time it takes for the motor to settle into the target (not including the InTargetTime).
- MotionSamples[3]: The time of the motion profile itself plus the time it takes for the motor to settle into the target (including the InTargetTime).

After power on or reset, all MotionSamples[] elements are set to -1, to indicate that no motion was performed yet.

Description	Value
<b>Mnemonic (keyword):</b>	MotionSamples
<b>CAN code:</b>	267
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MotionStat

MotionStat reports the status of the current motion. Each bit in MotionStat represents a motion state. It is possible that more than one bit will be on in certain cases. When the motor is not in motion MotionStat = 0. The table below shows the meanings of MotionStat bits:

Bit	Meaning
0	In motion
1	Waiting (during repetitive motion)
2	In repetitive stop (following StopRep command)
3	Stop requested
4	In acceleration
5	In deceleration
6	Waiting for smoothing to end
7	In ECAM stop (following StopECAM command)
8	In FIFO stop (following StopFIFO command)
9	In wait for Input (motion is suspended till rising edge at the user defined input)

Description	Value
<b>Mnemonic (keyword):</b>	MotionStat
<b>CAN code:</b>	32
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MotorCurr

MotorCurr is the total current to the motor (all phases combined). The value is reported in milliAmperes.

Description	Value
<b>Mnemonic (keyword):</b>	MotorCurr
<b>CAN code:</b>	8
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MotorOn

MotorOn: Enable/ Disable the motor.

MotorOn = 0 disables the motor.

MotorOn = 1 enables the motor.

When the motor is disabled power is not applied to the motor and it is not controlled by the driver.

The motor can be disabled internally by the driver due to a fault (see ConFlt).

When the user re-enables the motor ConFlt is cleared. If the actual state that caused the fault was not corrected (for example: over temperature caused a motor disable and the user tries to enable the motor before cooling), the motor will stay disabled.

Some commands and parameter changes are allowed only when the motor is disabled. See the attributes for each parameter.

Description	Value
<b>Mnemonic (keyword):</b>	MotorOn
<b>CAN code:</b>	130
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## MotorType

MotorType is the type of the motor that is connected to the controller. This variable determines how voltage is applied to the motor. Selecting the wrong MotorType may result in severe consequences.

The values that can be assigned to MotorType are:

Value	Motor Type
0	Unknown
1	DC brush
2	Voice coil
3	Linear DC brushless
4	Rotary DC brushless
5	Simulation

The meaning of each motor type is described below:

### Unknown

This is the default value of a new controller. No voltage will be applied to the power stage outputs before another valid motor type is set by the user.

### DC Brush

If a DC brush motor type is set the full voltage is applied to two power terminals. See the hardware manual to find out which terminals to use.

### Voice Coil

This motor type is the same as DC brush

### Linear/Rotary DC Brushless

The voltage is applied to three motor phases and commutation is used to determine the voltage that is applied to each motor power terminal. The difference between the types is actually in the configuration of the feedback and how motor poles are treated. The separate types are there to help with configuration in the PC suite.

See the hardware manual for correct phase connection.

### Simulation

This motor type is used for simulation during development. It allows generating simulated profiler, input and output behaviors without actually connecting a motor to the controller.

Description	Value
<b>Mnemonic (keyword):</b>	MotorType
<b>CAN code:</b>	50
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	7
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## OfflineALog

Description	Value
<b>Mnemonic (keyword):</b>	OfflineALog
<b>CAN code:</b>	620
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 45
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## OfflineBLog

Description	Value
<b>Mnemonic (keyword):</b>	OfflineBLog
<b>CAN code:</b>	621
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 45
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## OneOverTOn

1/T (OneOverT) is a process in which the motor velocity (using the main encoder feedback) is measured accurately, especially for low speeds.

The motor velocity, as reflected at the Vel[] parameter is derived from the encoder feedback by a simple derivative, as follows:

$$\text{Vel}[2] = (\text{PosK} - \text{PosK-1}) / T_s$$

( $T_s$  is the controller sampling time, typically with Agito's controllers:  $1/16384=61(s)$ ).

This creates a reading with poor resolution of  $1/T_s$  (typically 16384).

As a result, if the motor velocity is, for example, 10000 [counts/sec], the Vel[2] will show values of 0 and 16384.

This is, of course, not enough for accurate velocity measurement.

Indeed, Vel[1] is a filtered value of Vel[2], and Vel[3] is a moving average filter of Vel[2], but still, they do not provide an accurate and immediate velocity sensing, especially at low speeds.

At this point, the 1/T mechanism provides an alternative method to measure the motor velocity from the encoder feedback.

Generally speaking, the idea is to measure the time duration of the encoder input pulses. This is done between changes at the encoder input, using a timer that is fed by a very fast frequency clock. Now, if we have the time duration (let's call it  $T$ ) of, for example, 1 encoder count, we can calculate the accurate velocity by:

$$\text{Vel} = 1/T.$$

To enable accurate and valid measurement both at high and low speeds, the user can control the number of encoder input pulses (whose time period is measured) and the frequency that is used to measure this time period.

Higher frequencies will provide better accuracy for the measurement, but the time can overflow at low speeds (in case of overflow, the velocity reading is set to 0).

Higher number of encoder pulses will provide better accuracy, but the reading will be less immediate and the time can also reach overflow.

The velocity that is calculated using the 1/T mechanism is reported at Vel[4].

Relevant parameters:

OneOverTOn:

Set to 0 to disable the 1/T measurement (Vel[4] reports 0).

Set to 1 to enable the 1/T measurement.

OneOverTGap:

Defines the number of encoder pulses that are used to measure the time period.

The actual number of pulses is  $2 \times \text{OneOverTGap}$ .

Note that a value of 4 (or multiplication of 4) for the actual number of pulses will provide a more accurate velocity reading because it will not be affected by the shift between the A and B encoder signals (which are not always shifted exactly by 90°).

OneOverTFreq:

Defines the frequency that is used to measure the time duration.

The actual frequency, in Hz, is given by:

$1/T \text{ frequency} = \text{SYSTEM\_CLOCK} / 2\text{OneOverTFreq}$

As a result, the equation that is used to calculate the 1/T reading is:

$\text{Vel}[4] = \text{Number of pulse [counts]} / \text{Measured time [sec]}$

$\text{Vel}[4] = 2\text{OneOverTGap} / (\text{TimerValue} * \text{TimerPeriod})$

$\text{Vel}[4] = 2\text{OneOverTGap} * \text{Timer Frequency} / \text{TimerValue}$

$\text{Vel}[4] = (2\text{OneOverTGap} * \text{SYSTEM\_CLOCK} / 2\text{OneOverTFreq}) / \text{TimerValue}$

$\text{Vel}[4] = \text{SYSTEM\_CLOCK} / \text{TimerValue} * (2\text{OneOverTGap} / 2\text{OneOverTFreq})$

Note that the 1/T velocity reading is valid only for incremental encoder input.

Note that the 1/T velocity reading (reported at Vel[4]) is used only for reporting (data recording) and is not used as part of the closed control loop filters.

Description	Value
<b>Mnemonic (keyword):</b>	OneOverTOn
<b>CAN code:</b>	187
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Vel\[\]](#)

## OpenLoopCurr

Description	Value
<b>Mnemonic (keyword):</b>	OpenLoopCurr
<b>CAN code:</b>	145
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## OpenLoopOn

If we set OpenLoopOn=1 will switch to open loop mode. In this mode, we can inject current or voltage command accordingly.

Description	Value
<b>Mnemonic (keyword):</b>	OpenLoopOn
<b>CAN code:</b>	144
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

AopenLoopOn=1

**See Also:**

[OpenLoopVolt](#), [OpenLoopCurr](#)

## OpenLoopVolt

Description	Value
<b>Mnemonic (keyword):</b>	OpenLoopVolt
<b>CAN code:</b>	146
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[OpenLoopOn](#),

## OperationMode

OperationMode determines what control loops will be activated. The values for OperationMode are:

Value	Operation Mode
1	Current control only
2	Velocity control
3	Position control

### Current control

In this mode only the current is controlled. The value of the current reference (= current command) is taken from the designated analog input.

### Velocity control

In this mode the velocity and current are controlled. The value of the velocity reference (= velocity command) is taken from the designated analog input.

### Position control

This is the default motion mode. In this mode all the control loops are active. The position profiler generates a position command according to the motion requirements that were set by the user. The user can use the different motion modes to select whether to input a position target command or a velocity command.

Description	Value
<b>Mnemonic (keyword):</b>	OperationMode
<b>CAN code:</b>	78
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	4
<b>Default value:</b>	3
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

### See Also:

[MotionMode](#),

## PDFact

PDFact is a factor that multiplies the pulse direction input counting before it is used. PDFact and PDFactDen together determine the resulting generated reference as follows:

Reference = PDFact/ PDFactDen

PDFact can be positive or negative to enable a change in the direction of the signal, while PDFactDen is always positive.

Description	Value
<b>Mnemonic (keyword):</b>	PDFact
<b>CAN code:</b>	110
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-16,777,215
<b>Max value:</b>	16,777,215
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PDPos](#), [PDUsrUnits](#), [PDFiltFact](#), [PDFactDen](#)

## PDFactDen

PDFactDen divides the pulse direction input counting before it is used. PDFact and PDFactDen together determine the resulting generated reference as follows:

Reference = PDFact/ PDFactDen

PDFact can be positive or negative to enable a change in the direction of the signal, while PDFactDen is always positive.

Description	Value
<b>Mnemonic (keyword):</b>	PDFactDen
<b>CAN code:</b>	119
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	16,777,215
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PDPos](#), [PDUsrUnits](#), [PDFiltFact](#), [PDFact](#).



## PDFiltFact

Agito controllers support motion modes in which the desired motor position is defined by the Pulse/Direction input port.

Please refer to the documentation of the MotionMode parameter, PD\_DIRECT and PD\_INDIRECT motion modes.

The Pulse/direction input port value can be read using the parameter PDPos. It is equal to the number of input pulses (direction taken into account) multiplied by the parameter PDFact (refer to PDFact documentation).

In the PD\_DIRECT motion mode, PDPos is used to set directly the position reference (PosRef: desired position). However, in order to avoid large steps in PosRef (especially when PDFact is large), a first order filter is used to filter PDPos before it is assigned to PosRef:

PDFiltFact is the parameter which defines the bandwidth of this first order filter.

The relevant equations are:

$$\text{PosRefK} = (\text{PDPosK} * \text{PDFiltFact} + \text{PosRefK-1} * (64 - \text{PDFiltFact})) / 64$$

PDFiltFact gets values between 1 (slowest filter) to 64 (no filter at all).

Notes:

1. Future versions of Agito controllers will not input directly the filter's coefficient (which is not easy to calculate), but instead the user will define the desired filter frequency and the controller will automatically calculate the resulting filter coefficient.
2. The above described equations are only an illustration. Internally, the calculations are done relative to the initial PosRef and PDPos values, during the Begin function (command to start motion in PD\_DIRECT mode).

Description	Value
<b>Mnemonic (keyword):</b>	PDFiltFact
<b>CAN code:</b>	150
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	64
<b>Default value:</b>	3
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PDPos](#), [PDFact](#) and [MotionMode](#).

## PDPoS

PDPoS reports the pulse direction input position command reading in PD user units (PDUsUnits).

If PDUsUnits = 1 then PDPoS is in pulses. The value of PDPoS is 0 upon reset.

PDPoS counts the position between -2147483648 cnts and 2147483647. If the command input from PDPoS exceeds these limits the position command reading will roll.

PDUsUnits is used only to report the number of pulses that were counted. It is also possible to multiply the number of input pulses by a factor, and the result of the multiplication is used as the reference.

Reference =

The value of PDPoS displayed will be

PDPoS =

Example:

PDUsUnits = 5

(For example: 5 pulses represent 1mm motion, and the user wants to read the command in mm)

After 20 pulses are entered to the pulse / dir input:

PDPoS -> 4

After 3 more pulses are entered (a total of 23):

PDPoS -> 4

Description	Value
<b>Mnemonic (keyword):</b>	PDPoS
<b>CAN code:</b>	4
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	Pulse direction user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PDFact](#), [PDFactDen](#), [PDUsUnits](#), [PDFiltFact](#),

## PDUsrUnits

PDUsrUnits allows the user to read the pulse – direction command and its derivatives in units other than pulses. PDUsrUnits is the ration between the desired unit and the pulses. A full explanation of how user units work can be found in the beginning of this document.

Description	Value
<b>Mnemonic (keyword):</b>	PDUsrUnits
<b>CAN code:</b>	66
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	65,536
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

If the user wants to see the position command in mm, and every 5 pulses are equivalent to 1mm, set UsrUnits to 5.

The position command will now be received in mm.

### See Also:

[AuxUsrUnits](#), [UsrUnits](#)

## PDVel

PDVel reports the velocity of the pulse direction input in PDUrUnits/sec. This velocity is calculated as a derivative of PDPos.

Description	Value
<b>Mnemonic (keyword):</b>	PDVel
<b>CAN code:</b>	7
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	Pulse direction user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PDFact](#), [PDFactDen](#), [PDUrUnits](#), [PDFiltFact](#),

## PeakCL

Basically, the PeakCL is the peak current limit (in [mA]). The current command (CurrRef) will never exceed this value ( $\pm$ PeakCL is used to saturate the CurrRef).

The maximal allowed value for PeakCL (see the table above) is the maximal allowed current for this product.

Note that PeakCL, with sinusoidal commutation, refers to the peak value of the current sinusoidal signal, not to its RMS value.

However, the PeakCL parameter, together with the ContCL and PeakTime parameters, is used by the controller to implement the  $I^2t$  amplifier/motor power limitation scheme.

The maximal values of PeakCL, ContCL and PeakTime define the  $I^2t$  limitation scheme as required not to exceed the amplifier capabilities. However, the user can modify these parameters to define a new limitation scheme that will match, for example, the motor power limitations.

For example, a given product can have maximal values as follows:

PeakCL = 16000

ContCL = 8000

PeakTime = 1000

Which means that this product can drive up to 16000mA (16A) for a period of 1000ms (1 sec), and that its continuous current limitation is 8000mA (8A). This is the limitation of the product itself.

However, for a given motor/application, the user can set:

PeakCL = 5000

ContCL = 3000

PeakTime = 500

So not to exceed the power capabilities of the motor, or even of the amplifier if the rated cooling conditions are not provided by the application.

### How the amplifier/motor $I^2t$ power limitation scheme works?

This limitation uses the following parameters: ContCL (continuous current limit), PeakCL (peak current limit) and PeakTime (maximal allowed peak current time).

The limitation code always calculates  $I^2$  (using the overall motor current: MotorCurr) over the time ("integration" of  $I^2$ ). If it becomes higher than  $\text{ContCL}^2$ , the limitation is activated.

Once the limitation is activated, the CurrRef, typically limited (saturated) by  $\pm$ PeakCL, is now limited by  $\pm$ ContCL, so that actually the amplifier is now limited to provide current that is no more than ContCL.

At all times,  $I^2$  is calculated over the time, using a first order filter that emulates the heat dissipation at the motor (or better to say, the temperature of the motor). In this way,  $I^2$  is "integrated" over the time to estimate the temperature that is created within the

amplifier/motor. The first order filter's coefficient is calculated as a function of PeakTime, PeakCL and ContCL, so that:

Following a long time of no current to the motor, if a PeakCL current is now driven (PeakCL<sup>2</sup> is the input to the filter), for a period of PeakTime, the output of the filter will reach the ContCL<sup>2</sup> value, which will trigger the limitation and will limit the current to ContCL.

Of course, if a lower current (lower than PeakCL) is driven to the motor, a longer time will be available without triggering the limitation. Actually, if the current is equal or below ContCL, the limitation will be never activated.

Note that the availability of PeakCL current, for a period of PeakTime, is only after long time of no current to the motor (so the filter output is 0). In other cases (e.g.: a long time of  $0.9 \times \text{ContCL}$ ), the actual peak time will be (much) shorter, as the filter output will reach ContCL<sup>2</sup> much faster (this is equivalent to the motor's temperature behavior, as in this case the motor was already heated due to the non-zero current over the time).

Once  $I^2$  over the time goes below  $0.9 \times \text{ContCL}^2$ , the limitation is released. This is needed to create a hysteresis, so that the limitation will not go on/off/on/off very fast around the continuous current.

Note that all the parameters of this limitation can be modified on-the-fly.

Note that the limitation works only on the CurrRef saturation that is typically PeakCL. It has no effect on other limitations that are supported by the controller: Current limitations using analog inputs of fixed values (see: CurrLimMode).

Note that while the user can set values like PeakCL=100 and ContCL = 200 (i.e. Peak < Continuous or Peak=Continuous), such case is invalid, and the controller will internally use a different value for the continuous current, which is equal to PeakCL/2. Once the user will properly set both parameters, the controller will return to use the values of PeakCL and ContCL.

The limitation status is reflected at the StatReg parameter, bits 24, 25. In the PC Suite, you will see the "Other warn." LED going "orange" color when the limitation is activated.

**Note:**

This  $I^2t$  power limitation scheme is working only if the current control loop is activated (refer to the ControlMode parameter) or if an external amplifier is used and CurrRef is used to drive an analog output.

In the latter case, CurrRef is used in the equations of this limitation, instead of MotorCurr (see explanation above).

Description	Value
<b>Mnemonic (keyword):</b>	PeakCL
<b>CAN code:</b>	52
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ContCL](#), [PeakTime](#), [MotorCurr](#) and [StatReg](#).

## PeakTime

The PeakTime (Maximal time for peak current, in [msec]) is used, together with PeakCL and ContCL parameters, to define the behavior of the amplifier/motor I2t power limitation scheme. Please refer to the PeakCL keyword page for detailed description of the limitation scheme and the effect of PeakTime.

Description	Value
<b>Mnemonic (keyword):</b>	PeakTime
<b>CAN code:</b>	53
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	3,000
<b>Default value:</b>	500
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PeakCL](#), [ContCL](#), [MotorCurr](#) and [StatReg](#).



## PolePrs

PolePrs is the number of magnetic pole pairs per mechanical revolution in the controlled motor. This number, together with EncRes is used to determine the pattern of voltage applied to the motor phases. Please use the motor data sheet to enter the correct value of PolePrs.

For linear motor enter PolePrs = 1, EncRes = the number of encoder counts per electric cycle

Warning:

If PolePrs is wrong unexpected behavior will occur. This can result in severe damage to the controller, motor or any other system parts connected to the motor.

Description	Value
<b>Mnemonic (keyword):</b>	PolePrs
<b>CAN code:</b>	54
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	50
<b>Default value:</b>	4
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[EncRes](#).

## PopParam

PopParam is a user program low level language keyword. It will pop the last ("top") value in the numeric stack of the current thread and assign it to the requested parameter. The parameter to be assigned is indicated by using its complex CAN code.

Normally, the user does not need to be concerned with generating the code since the user program IDE environment on the PC Suite will automatically generate it during compilation.

Description	Value
<b>Mnemonic (keyword):</b>	PopParam
<b>CAN code:</b>	202
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Pos

Pos reports the main encoder position reading in user units (*UsrUnits*). If *UsrUnits* = 1 then Pos is in encoder counts. The value of Pos is 0 upon reset.

Pos counts the position between -2147483648 cnts and 2147483647. If the actual position of the motor exceeds these limits the position reading will roll. To prevent this from happening use the modulo function (ModRev).

With the motor off Pos can also be set by the user to a desired value.

Description	Value
<b>Mnemonic (keyword):</b>	Pos
<b>CAN code:</b>	2
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PosErr

PosErr returns the value of the position error in user units (UsrUnits). The position error is the difference between the position command (PosRef) and the actual position. If the value of PosErr exceeds the maximum allowed position error as defined in MaxPosErr the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	PosErr
<b>CAN code:</b>	18
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

### Example:

If PosRef = 1000 and Pos = 990 then PosErr = 10

### See Also:

[PosRef](#), [Pos](#),

## PosFiltDef

Please refer to the [PosFiltOn](#) keyword page for detailed description of the position bi-quad filters function.

Description	Value
<b>Mnemonic (keyword):</b>	PosFiltDef
<b>CAN code:</b>	123
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosFiltOn](#),

## PosFiltOn

Agito's controllers include some bi-quad filters as part of the closed loop control filters. At the moment, a single bi-quad filter is supported at the position control and two bi-quad filters are supported at the velocity control (refer to VelFiltOn[] keyword page). The bi-quad filter at the position control is actually a filter that is applied on the position reference is used as a vibrations suppression filter. It is typically a notch filter (at the frequency to be rejected), but actually any type of filter can be used (as described below).

Filters are defined using the PosFiltOn[] and PosFiltDef[] parameters. PosFiltOn[N] enables ("1") or disables ("0") the filter number N. When the filter is disabled, it actually means that the filter is equal to 1. At the moment, as explained above, only PosFiltOn[1] is used, to enable/disable the vibrations suppression filter.

PosFiltDef[] is used to define the filter type and its parameters: PosFiltDef[] is divided into groups of 5 elements (currently only 1 group is used). This means: PosFiltDef[1] to PosFiltDef[5] refer to the first filter. PosFiltDef[6-10] refer to the second filter and so on. The first element of each group defines the filter's type. The rest 4 elements define the filter characteristics (frequency, damping, ...) as relevant for each type of filter (not all of them use all the 4 elements). The list of supported filter's types and their related parameters are described below.

Upon completing the definitions of PosFiltOn[] and PosFiltDef[] (and actually also any change of VelFiltOn[] or VelFiltDef[]), the user must send the CalcFilters command, so that this input data will be used to calculate the actual filter coefficients. These coefficients are stored at the PosFilt[] array (5 elements per each filter).

The PosFilt[] is a read only array.

The structure of the internal implementation of the filter is as follows (for the first filter, that is stored at PosFilt[1-5]):

$$YK = (XK * PosFilt[1] + XK-1 * PosFilt[2] + XK-2 * PosFilt[3] - YK-1 * PosFilt[4] - YK-2 * PosFilt[5]) / 65536$$

Where X is the input to the filter and Y is its output.

Notes:

Upon any change of PosFiltOn[] and PosFiltDef[], the controller will not allow enabling of the axis until a successful CalcFilters operation. This is to ensure that the control loop is using the filters defined by the user (at PosFiltOn[], PosFiltDef[]).

List of supported filters and their characteristic parameters (as defined within PosFiltDef[]): In the description below, the numeric value that represents each type of filter is the value to put at PosFiltDef[1] (the first element of the group of 5) and par1 to par4 are actually the values at PosFiltDef[2-5].

//

```
//      Note: All frequency parameters (poles, zeros ...) that are described below are
//      actually provided as Hz*100, not Hz. This is to enable frequencies with
//      fractions.
//
//      1 - LPF1: 1st order Low Pass Filter the par1 is the pole frequency in
//      Hz, par2, 3 and 4 aren't applicable.
//
//      2 - LPF2: 2nd order Low Pass Filter, par1 is the pole frequency in
//      Hz, par2 is the pole damping ratio. Par3 and 4 aren't applicable.
//
//      3 - LPF3: 2nd order Low Pass Filter with zero,
//      par1 is the pole frequency in
//      Hz, par2 is the pole damping ratio and par3 is the zero frequency
//      in Hz. par4 isn't applicable.
//
//      4 - LDLG2: 2nd order Lead/Lag filter
//      par1 is the lead first zero frequency, par2
//      is the lead second zero, par3 is the leg first pole and
//      par4 is the leg second pole.
//
//      5 - LDLG1: 1st order Lead/Lag filter
//      par1 is the lead zero frequency and par2 is the leg pole.
//
//      6 - LDLG1FP: 1st order Lead/Lag filter defined by the frequency at the
//      phase peak/min and the phase level.
//      par1 is the frequency at the phase's peak/min in Hz,
//      par2 in the required phase peak in degrees.
//
//      7 - LDLG2FP: 2nd order Lead/Lag filter defined by the frequencies at
//      the phases peak/min and the phases level (Approximately)
//      par1 is the 1st frequency at the phase peak/min in Hz,
//      par2 in the required phase peak in degrees at the 1st frequency.
//      par3 is the 2nd frequency at the phase peak/min in Hz,
//      par4 in the required phase in degrees at the 2nd frequency.
//
//      8 - NOTCH: Notch filter
//      par1 is the notch frequency in Hz, par2 is
//      the notch depth in dB and par 3 is the Notch width in Hz.
//      par4 isn't applicable.
//
//      9 -CLDLG: Complex Lead/Leg filter.
//      par1 - frequency of the complex zero in Hz, par2 damping
//      ratio of the complex zero, par3 is the frequency of the
//      pole in Hz and par4 is the damping ratio of pole.
//
```

Description	Value
<b>Mnemonic (keyword):</b>	PosFiltOn
<b>CAN code:</b>	124
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 1
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosFiltDef\[\]](#), [VelFiltOn\[\]](#), [VelFiltDef\[\]](#), [VelFiltDef\[\]](#) and [CalcFilters](#).



## PosGain

Agito controllers implement a position over velocity control filter scheme.

The position control filter consists of gain (this is PosGain) and several bi-quad filters. The actual number of the filters may vary according to product.

PosGain[] is an array, to serve the gain scheduling mechanism. At any time, a given element of PosGain[] is used for the control filter calculations, based on the current decision of the gain scheduling mechanism.

The default element is PosGain[1].

The user may set all elements of PosGain[] to be equal, so the same gain value will be used with any decision of the gain scheduling mechanism. Setting different values to PosGain[] elements allows different gains to be used for each case.

The relevant equations in the controller are (assuming no position bi-quad filter is used):

$\text{PosErr} = \text{PosRef} - \text{Pos}$ .

$\text{VelRef} = \text{PosErr} * \text{PosGain}[\text{As Selected By Scheduling}] + (\text{PosRef} - \text{PosRefPrev}) * \text{SAMPLE\_FREQUENCY}$

The second element is a built-in velocity feed-forward that is required for proper control equations.

It is important to note that following these equations, VelRef itself is limited by  $\pm \text{MaxVel}$  (Maxvel is a controller parameter).

Description	Value
<b>Mnemonic (keyword):</b>	PosGain
<b>CAN code:</b>	100
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	32,768
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VelGain](#), [VelKi](#), [AccFFW](#), [PosFiltDef](#), [VelFiltDef](#), [MaxVel](#), [ScheduleMode](#).

## PosPosFlag

Description	Value
<b>Mnemonic (keyword):</b>	PosPosFlag
<b>CAN code:</b>	328
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PosPosTh

Description	Value
<b>Mnemonic (keyword):</b>	PosPosTh
<b>CAN code:</b>	329
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PosRef

PosRef returns the position reference generated by the internal profiler in user units (UsrUnits).  
The value of PosRef is fed to the position control loop.

Description	Value
<b>Mnemonic (keyword):</b>	PosRef
<b>CAN code:</b>	24
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PowerSupply

PowerSupply is used for protection against disconnecting the main power. The protection will be activated according to the type of power supply that is set in PowerSupply.

Single phase: 1

Three phases: 3

See the hardware manual for the correct connection of the power supply.

Description	Value
<b>Mnemonic (keyword):</b>	PowerSupply
<b>CAN code:</b>	401
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	3
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgBreaks

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgBreaks
<b>CAN code:</b>	294
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 3
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgCallDepth

Inquires the empty spaces at the program calls stack of the specified thread.

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgCallDepth
<b>CAN code:</b>	277
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgCallStack

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgCallStack
<b>CAN code:</b>	276
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	50
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## ProgClrCall

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgClrCall
<b>CAN code:</b>	275
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgClrExp

ProgClrExp is a user program low level language keyword. It is used to clear the numeric stack of the current thread. It is recommended to begin new programs by using ProgClrExp.  
Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgClrExp
<b>CAN code:</b>	203
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 9
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgExpStack](#), [ProgExpDepth](#).

## ProgError

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgError
<b>CAN code:</b>	199
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgEventEn

Activate ("1") or disables ("0") the handling of user program events. When disabled ("0") all pending events are cleared and events are not handled/processed at all. This includes also the sensing of events.

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventEn
<b>CAN code:</b>	524
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgEventGEn

Globally enables ("1") or disables ("0") the servicing of all events. ProgEventGEn, when "0", does not disable the sensing of events, and events are still sensed and possibly pending, to be serviced when enabled.

Please refer to the User Program Language Manual for more information.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventGEn
<b>CAN code:</b>	526
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgEventMask

Defines a bitwise mask to apply on the user defined event trigger parameter. The mask is also applied on the trigger value.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventMask
<b>CAN code:</b>	521
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgEventOn

Activate ("1") or disables ("0") the handling of user program events. When disabled ("0") all pending events are cleared and events are not handled/processed at all. This includes also the sensing of events.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventOn
<b>CAN code:</b>	527
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgEventPar](#), [ProgEventStat](#), [ProgEventType](#),

## ProgEventPar

Defines (using Complex CAN Code) which controller parameter to use for the triggering of this event. If ProgEventPar[EventNumber] is set to 0 (or to a non-valid Complex CAN code), this event will not be sensed and will not be handled.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventPar
<b>CAN code:</b>	520
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgEventOn](#), [ProgEventStat](#), [ProgEventType](#),



## ProgEventStat

Reports the state of this event. "0" for waiting for trigger, "1" for pending for service (triggered) and "2" for in service. Note that this mean that while a given event is being serviced, it can't not be triggered again, till servicing is completed (returning from the event function using the Return keyword).

This parameter is R/W, so user can clear a pending event – only the value of 0 can be written to this parameter).

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventStat
<b>CAN code:</b>	525
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgEventOn](#), [ProgEventPar](#),

## ProgEventType

Defines the type of the trigger (rising edge, equal, not equal...). Note that the four parameters to define a trigger for an event are very like the definition of a trigger for data recording.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventType
<b>CAN code:</b>	522
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	8
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgEventVal

Define the value to be used for the trigger detection.

Description	Value
<b>Mnemonic (keyword):</b>	ProgEventVal
<b>CAN code:</b>	523
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgExpDepth

ProgExpDepth is a user program low level language keyword. ProgExpDepth returns the highest full location of the relevant numeric stack. If the stack is empty it will return -1. If there is one value in the stack the highest full location is 0, etc...

Description	Value
<b>Mnemonic (keyword):</b>	ProgExpDepth
<b>CAN code:</b>	205
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 9
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgClrExp](#), [ProgExpStack](#).

## ProgExpStack

ProgExpStack is a user program low level language keyword. It is used to read the top number of the numeric stack without popping it. This keyword is most useful for debug.

Description	Value
<b>Mnemonic (keyword):</b>	ProgExpStack
<b>CAN code:</b>	204
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 9
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	51
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgClrExp](#), [ProgExpDepth](#).

## ProgFunc

ProgFunc keyword is used as a label in user programs. In high level the following text can be used, for example:

```
...  
AProgFuncCall,1  
...  
AProgFunc[1]  
// The contents of function 1  
AReturn
```

When the line AProgFuncCall,1 is reached, the program execution jumps to the location of the label keyword ProgFunc[1]. Return will cause a jump back to the user program that will continue execution on the next line.

Use multiple ProgFunc[] labels for multiple functions.

**Note:**

Use ProgHalt at the end of the program if your program is not an endless loop. Otherwise execution will continue into the first function and the “return” keyword will cause an error.

Description	Value
<b>Mnemonic (keyword):</b>	ProgFunc
<b>CAN code:</b>	431
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	20
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgFuncCall

When the line AProgFuncCall,1 is reached, the program execution jumps to the location of the label keyword ProgFunc[1]. Return will cause a jump back to the user program that will continue execution on the next line.

Use multiple ProgFunc[] labels for multiple functions.

Note:

Use ProgHalt at the end of the program if your program is not an endless loop. Otherwise execution will continue into the first function and the “return” keyword will cause an error.

Description	Value
<b>Mnemonic (keyword):</b>	ProgFuncCall
<b>CAN code:</b>	430
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	20
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ProgFunc](#), [ProgHalt](#).

## ProgHalt

“ProgHalt[Thread no.]” will halt the thread number. Halting is not the same as resetting the program. If ProgRun is entered again for the same thread it will continue from the point it was halted in.

Description	Value
<b>Mnemonic (keyword):</b>	ProgHalt
<b>CAN code:</b>	197
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

AProgHalt[3]

This command will halt thread 3.

### See Also:

[ProgRun](#), [ProgReset](#), [ProgResetAll](#),



## ProgHaltAll

Halt all the currently active user program threads.

Description	Value
<b>Mnemonic (keyword):</b>	ProgHaltAll
<b>CAN code:</b>	278
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgHaltThis

Description	Value
<b>Mnemonic (keyword):</b>	ProgHaltThis
<b>CAN code:</b>	258
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgInfo

Inquires the list of information strings that are included with the user program: CRC value, Date, CUP File name and the text information (see the "#information" compiler directive)

Description	Value
<b>Mnemonic (keyword):</b>	ProgInfo
<b>CAN code:</b>	297
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgLine

Description	Value
<b>Mnemonic (keyword):</b>	ProgLine
<b>CAN code:</b>	1021
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgPointer

Description	Value
<b>Mnemonic (keyword):</b>	ProgPointer
<b>CAN code:</b>	279
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgPriority

Description	Value
<b>Mnemonic (keyword):</b>	ProgPriority
<b>CAN code:</b>	296
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	10
<b>Default value:</b>	1
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgReset

Description	Value
Mnemonic (keyword):	ProgReset
CAN code:	295
Type:	Command
Access:	Read only
Allowed in motion:	Yes
Allowed with motor on:	Yes
Array with index range of:	1 : 8
Save to flash:	No
Axis related:	No
Min value:	0
Max value:	0
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

[ProgRun](#), [ProgResetAll](#),

## ProgResetAll

ProgResetAll will stop any running user program threads and reset all the pointers and stacks.

Description	Value
<b>Mnemonic (keyword):</b>	ProgResetAll
<b>CAN code:</b>	192
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ProgRun](#).



## ProgRun

“ProgRun[Thread no.], Task no.” will run the required task as the required thread number. For example:

AProgRun[3],5

Runs task 5 as thread 3.

To run the “main” program that starts at the beginning of the file use “-1” as the task number.

Description	Value
<b>Mnemonic (keyword):</b>	ProgRun
<b>CAN code:</b>	198
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-1
<b>Max value:</b>	30
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgSingle

ProgSingle is a user program related command that is normally issued by the PCsuite.

AProgSingle[1],0

Executes the next line of thread 1 and halts. This is like a debugger “step into”.

AProgSingle[1],1

Is equivalent to debugger “step over” for internal wait loops.

Description	Value
<b>Mnemonic (keyword):</b>	ProgSingle
<b>CAN code:</b>	191
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgStat

Inquires the status of the specified program thread:

- 1: No user program in the controller
- 0: Not running
- 1: Running

Description	Value
<b>Mnemonic (keyword):</b>	ProgStat
<b>CAN code:</b>	259
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-1
<b>Max value:</b>	1
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgStatAll

Description	Value
<b>Mnemonic (keyword):</b>	ProgStatAll
<b>CAN code:</b>	298
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-1
<b>Max value:</b>	2
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ProgTask

ProgTask keyword is used as a label in user programs. The task is called by using AProgRun[thread],task no. In high level the following text can be used

This program runs the task after label AProgTask[task no] until ProgHalt using the assigned thread number.

**Note:**

If halt is not used the code will continue linearly to the next line in the file.

Description	Value
Mnemonic (keyword):	ProgTask
CAN code:	433
Type:	Command
Access:	Read only
Allowed in motion:	Yes
Allowed with motor on:	Yes
Array with index range of:	1 : 30
Save to flash:	No
Axis related:	No
Min value:	1
Max value:	30
Default value:	1
User units:	No user units
Implementation status:	Implemented

**Example:**

```
...  
AProgRun[thread],task no  
...  
AProgTask[1]  
// The contents of task 1  
AProgHalt
```

**See Also:**

## ProtectMask

ProtectMask can be used to mask some hardware protections. Protections that are masked will not be activated. Most hardware protections are non maskable.

The protections that can be masked:

To mask the “main encoder disconnected” protection  $\text{ProtectMask} = \text{ProtectMask} \mid 0x0004$

To mask the “auxiliary encoder disconnected” protection  $\text{ProtectMask} = \text{ProtectMask} \mid 0x0008$

The easiest way to mask protections is to use the PC suite’s Configuration -> Protections window and check the required mask.

Description	Value
<b>Mnemonic (keyword):</b>	ProtectMask
<b>CAN code:</b>	97
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,047
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PStatInterval

Description	Value
<b>Mnemonic (keyword):</b>	PStatInterval
<b>CAN code:</b>	482
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	2
<b>Max value:</b>	10,000
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PStatOn

Description	Value
<b>Mnemonic (keyword):</b>	PStatOn
<b>CAN code:</b>	480
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## PStatParams

Description	Value
<b>Mnemonic (keyword):</b>	PStatParams
<b>CAN code:</b>	483
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 20
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PStatPort

Description	Value
<b>Mnemonic (keyword):</b>	PStatPort
<b>CAN code:</b>	481
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	3
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PTPKeepMoving

Description	Value
Mnemonic (keyword):	PTPKeepMoving
CAN code:	625
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	Yes
Min value:	0
Max value:	1
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

## PushConstant

PushConstant is a user program low level language keyword. It is used to push the value of a constant into the numeric stack of the current user program thread.

Normally, the user does not need to be concerned with generating the code since the user program IDE environment on the PC Suite will automatically generate it during compilation.

Description	Value
<b>Mnemonic (keyword):</b>	PushConstant
<b>CAN code:</b>	201
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PushParam

PushParam is a user program low level language keyword. It is used to push the value of a parameter into the numeric stack of the current user program thread. The parameter to be pushed is indicated by using its complex CAN code.

Normally, the user does not need to be concerned with generating the code since the user program IDE environment on the PC Suite will automatically generate it during compilation.

Description	Value
<b>Mnemonic (keyword):</b>	PushParam
<b>CAN code:</b>	200
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## PwrTemp

PwrTemp returns the temperature in Celsius measured on the power generating part of the product.

Description	Value
<b>Mnemonic (keyword):</b>	PwrTemp
<b>CAN code:</b>	38
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[MaxPwrTemp](#)

---

## RecData

---

RecData is an array that holds all the information pertaining the latest recording. This data is best viewed as a graph using agito's PC Suite. The information below is mainly useful for a user that wants to write their own software to analyze the data.

Use RecUpload to receive a comma delimited list of the values in RecData. Note that RecUpload does not only upload the values as they are but also converts them if needed to values that are useful to the user. Some of the conversions use internal ratios, so the user cannot repeat them. The values of RecData can be accessed individually by querying RecData[n]. This is not recommended for the obvious reason that it is inconvenient, but also because RecUpload will convert the raw data to user data.

#### RecData Header

The first 40 values of RecData (RecData[1] : RecData[40]) contain information about the recorded data. The main purpose of most of the values is to serve as a mirror for the condition in the moment the recording started. They are used internally in the recording process and have little value in themselves.

RecData Index	Meaning
1	The expected number of values to be recorded (Number of parameters * RecLength)
2	RecParam[1]
3	RecParam[2]
4	RecParam[3]
5	RecParam[4]
6	Reserved
7	Reserved
8	Reserved
9	Reserved
10	Requested recording length (RecLength)
11	Recording gap in sample times (RecGap)
12	Trigger source complex CAN code (RecTrigSrc)
13	Trigger mask (RecTrigMask)
14	Trigger type (RecTrigType)
15	Trigger value (RecTrigVal)
16	Control loop frequency
17	Index of the 1st element of the recording
18	Index of the trigger
19	Last index in the recording
20	For internal use only (mirror of unit's conversion or indication that internal conversion is needed for recorded variable 1)



21	For internal use only (mirror of unit's conversion or indication that internal conversion is needed for recorded variable 2)
22	For internal use only (mirror of unit's conversion or indication that internal conversion is needed for recorded variable 3)
23	For internal use only (mirror of unit's conversion or indication that internal conversion is needed for recorded variable 4)
24	Constant (internal use)
25	Number of points recorded
26	Reserved
27	Reserved
28	Reserved
29	Reserved
30	Reserved
31	Reserved
32	Reserved
33	Reserved
34	Reserved
35	Reserved
36	Reserved
37	Reserved
38	Reserved
39	Reserved
40	Reserved

#### Recorded Values

The recorded values are stored in locations 41 to 8041 of RecData. The values for each sampling are in consecutive locations. If, for example, three parameters are recorded (without trigger):  
 $\text{RecData}[41] = \text{Param1}(T1)$

RecData[42] = Param2(T1)

RecData[43] = Param3(T1)

RecData[44] = Param1(T2)

RecData[45] = Param2(T2)

RecData[46] = Param3(T2)

And so on.

If a trigger is required the data is saved cyclically in RecData.

The recorded values are originally saved in internal units. They are converted to user units by RecUpload.

Description	Value
<b>Mnemonic (keyword):</b>	RecData
<b>CAN code:</b>	238
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 16540
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecParam](#), [RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecGap

RecGap determines how many sample times elapse between one recording time and the next.

Example:

To record a single parameter in the highest possible resolution set RecGap = 1. This way, the parameter is recorded every sampling time. Since the maximum length of the recording is 8,000 it is possible to record this parameter for no more than 8,000 sampling times. With a 16 KHz sample rate this is 0.5 sec.

If a longer recording time is required RecGap should be changed accordingly. RecGap = 4 will capture 2 seconds, but could miss details since it only captures the value of the parameter every 4 sample times.

Description	Value
<b>Mnemonic (keyword):</b>	RecGap
<b>CAN code:</b>	242
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[RecParam](#), [RecLength](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecLength

RecLength determines how many points will be recorded for each recorded parameter. If only one parameter is recorded RecLength can be up to 8,000 points. If 4 Parameters are recorded then RecLength can be no more than 2,000 because there are 4 values saved for each point.

Description	Value
<b>Mnemonic (keyword):</b>	RecLength
<b>CAN code:</b>	241
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	16,500
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecParam](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecParam

RecParam array holds the complex CAN codes of the parameters to record. There can be up to 4 parameters. All the assigned parameters until a zero is encountered will be recorded. To record a single parameter enter the complex CAN code of the parameter that should be recorded in RecParam[1] and enter 0 in RecParam[2]. The values of RecParam[3] and [4] will be ignored.

Description	Value
<b>Mnemonic (keyword):</b>	RecParam
<b>CAN code:</b>	240
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 8
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecStart

RecStart command will begin the recording process according to all the current recording parameters. Changing any parameters after the recording started will not affect the current recording.

Description	Value
<b>Mnemonic (keyword):</b>	RecStart
<b>CAN code:</b>	248
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecStat

RecStat will return the status of the current recording:

Value	Status
0	Data not valid (default after power on)
1	Filling pre-trigger data (waiting until enough data is acquired according to RecTrigPos)
2	Buffering and waiting for trigger
3	Trigger detected
4	Recording complete
5	Recording stopped
6	Recording stopped before trigger

Description	Value
<b>Mnemonic (keyword):</b>	RecStat
<b>CAN code:</b>	249
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	5
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecStop

RecStop stops the current recording process regardless of whether the data is complete. The actual length of the recording is updated in the RecData header. Other data may be incomplete or erroneous.

Description	Value
<b>Mnemonic (keyword):</b>	RecStop
<b>CAN code:</b>	250
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecTrigMask](#), [RecTrigForce](#).



## RecTrigForce

After the recording started, the user may decide not to wait for the trigger condition to be fulfilled, but force the recording to begin immediately. RecTrigForce will overrule the trigger condition detection and make the recording continue as if a trigger were detected.

Note that if the pre-trigger filling is not yet finished some additional time will pass until the buffer is filled and only then the trigger will be forced.

Description	Value
<b>Mnemonic (keyword):</b>	RecTrigForce
<b>CAN code:</b>	252
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecTrigMask

The purpose of RecTrigMask is to enable triggering on a single bit or a set of bits. The default value of RecTrigMask is -1, which is the decimal value equivalent to 0xFFFFFFFF. This means that no bits are masked, and all the bits of the trigger source parameter are considered when the trigger condition is tested.

To leave a single bit on RecTrigMask should be set to the decimal equivalent of a signed long hexadecimal value with only the required bit on and the other bits off. RecTrigMask and the trigger source parameter go through a bitwise and to produce the value that is used for trigger checking.

Description	Value
<b>Mnemonic (keyword):</b>	RecTrigMask
<b>CAN code:</b>	251
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

MotionStat bits indicate the status of the motion. Bit 5 of MotionStat is "in deceleration". To start a recording when the motor start decelerating we should use only bit 5 as trigger.

RecTrigSrc = 32 (trigger source is MotionStat)

RecTrigTyp = 3 (trigger type = not equal)

RecTrigVal = 0 (trigger value = 0)

To calculate RecTrigMask: to mask bit 5 the needed hex value is 0x0000 0020. The equivalent decimal value is 32.

RecTrigMask = 32

To return to normal trigger mode return RecTrigMask = 0.

### See Also:

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigForce](#).

## RecTrigPos

RecTrigPos is the location of the trigger in the recording. The value of RecTrigPos is given in percent. RecTrigPos allow the user to see recorded data of what happened immediately before the trigger and not only what happened after the trigger was detected.

Description	Value
<b>Mnemonic (keyword):</b>	RecTrigPos
<b>CAN code:</b>	247
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	100
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

For a recording of 5 seconds, if RecTrigPos = 20 then the recorded data will include information for 1 second before the trigger was detected and 4 seconds after the trigger was detected. Note that the recording mechanism does not start to look for a trigger until at least one second has elapsed since the beginning of the recording.

### See Also:

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecTrigSrc

RecTrigSrc holds the complex CAN code of the recording trigger source. This is the variable that is used as a trigger to the recording process.

Description	Value
<b>Mnemonic (keyword):</b>	RecTrigSrc
<b>CAN code:</b>	243
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

To use the position of axis A as the source of the trigger for the recording RecTrigSrc = 2.  
It is possible to use the position as trigger and record it at the same time.

### See Also:

[RecLength](#), [RecGap](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecTrigTyp

RecTrigTyp determines the type of trigger to be used in the recording:

Value	Trigger type
1	Greater than
2	Equal
3	Not equal
4	Less than
5	Rising edge
6	Falling edge
7	Manual (force trigger)

Description	Value
<b>Mnemonic (keyword):</b>	RecTrigTyp
<b>CAN code:</b>	245
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	8
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

To start the recording when the position of axis A passes through location 1000 with a rising edge, the trigger settings for the recording should be:

RecTrigSrc = 2 (The source of the trigger is axis A position)

RecTrigVal = 1000 (The value assigned to the trigger is 1000)

RecTrigTyp = 5

### See Also:

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecTrigVal

RecTrigVal is the value assigned to the trigger for the recording. This is the value that the trigger source parameter is compared to when determining if a trigger was detected.

Description	Value
<b>Mnemonic (keyword):</b>	RecTrigVal
<b>CAN code:</b>	246
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecUpload](#), [RecTrigTyp](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RecUpload

The command RecUpload will upload the contents of RecData[] array. It is best not to use this command directly since the contents of RecData needs interpreting. The best way to upload the recorded data is by using Agito PC Suite.

If RecUpload is sent in RS232 communication the values of RecData are returned as a comma delimited list. In CAN each value is uploaded as a 5 byte message containing 4 bytes long value and 1 byte ASCII comma.

Description	Value
<b>Mnemonic (keyword):</b>	RecUpload
<b>CAN code:</b>	244
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RecLength](#), [RecGap](#), [RecTrigSrc](#), [RecTrigTyp](#), [RecTrigVal](#), [RecTrigPos](#), [RecStart](#), [RecStat](#), [RecStop](#), [RecTrigMask](#), [RecTrigForce](#).

## RefOffsetSamp

Agito's controllers support adding an offset to the position profiler output (position reference, PosRef), during motion. This offset is added step by step, over a predefined period of time. The position reference offset is generally used to synchronize two axes (or to create a desired phase alignment between two synchronized axes).

The position reference offset is defined by two parameters. The RefOffsetStep defines the amount of offset (in [counts]) to add to PosRef each controller sampling. The RefOffsetSamp defines the number of samples in which this step will be added.

RefOffsetSamp is cleared to zero by the controller whenever the axis is not in motion.

While in motion, and if the user wants to add an offset to PosRef, the user first needs to set the value of RefOffsetStep (can be positive or negative) and then to set the value RefOffsetSamp. Once the (non-zero) value is written to RefOffsetSamp, the controller starts to add the offset, step by step, each control sample, while decrementing RefOffsetSamp by 1 each sample. Once RefOffsetSamp reaches a value of zero, the controller stops the process.

In order to support slow rates of the offset addition, the RefOffsetStep is scaled by  $1/T_s$ , with  $T_s$  the controller sampling time. Typically with Agito's controllers,  $1/T_s = 16384$ .

As a result, RefOffsetStep of 16384 actually means to add 1 encoder count each sample. And RefOffsetSamp of 20480 means to add 1.25 encoder counts each sample.

The overall offset that is added to PosRef is equal to:  $\text{RefOffsetStep} * \text{RefOffsetSamp} / 16384$ . The offset addition mechanism works only when the axis is in motion and not during a stopping motion segment.

Description	Value
<b>Mnemonic (keyword):</b>	RefOffsetSamp
<b>CAN code:</b>	165
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RefOffsetStep](#), [PosRef](#) and [MotionStat](#).



## RefOffsetStep

Agito's controllers support adding an offset to the position profiler output (position reference, PosRef), during motion. This offset is added step by step, over a predefined period of time.

The position reference offset is generally used to synchronize two axes (or to create a desired phase alignment between two synchronized axes).

The position reference offset is defined by two parameters. The RefOffsetStep defines the amount of offset (in [counts]) to add to PosRef each controller sampling. The RefOffsetSamp defines the number of samples in which this step will be added.

RefOffsetSamp is cleared to zero by the controller whenever the axis is not in motion.

While in motion, and if the user wants to add an offset to PosRef, the user first needs to set the value of RefOffsetStep (can be positive or negative) and then to set the value RefOffsetSamp. Once the (non-zero) value is written to RefOffsetSamp, the controller starts to add the offset, step by step, each control sample, while decrementing RefOffsetSamp by 1 each sample. Once RefOffsetSamp reaches a value of zero, the controller stops the process.

In order to support slow rates of the offset addition, the RefOffsetStep is scaled by  $1/T_s$ , with  $T_s$  the controller sampling time. Typically with Agito's controllers,  $1/T_s = 16384$ .

As a result, RefOffsetStep of 16384 actually means to add 1 encoder count each sample. And RefOffsetSamp of 20480 means to add 1.25 encoder counts each sample.

The overall offset that is added to PosRef is equal to:  $\text{RefOffsetStep} * \text{RefOffsetSamp} / 16384$ .  
The offset addition mechanism works only when the axis is in motion and not during a stopping motion segment.

Description	Value
<b>Mnemonic (keyword):</b>	RefOffsetStep
<b>CAN code:</b>	166
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-163,840
<b>Max value:</b>	163,840
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RefOffsetSamp](#), [PosRef](#) and [MotionStat](#).

## RegenOff

RegenOff is the bus voltage in millivolts that deactivates the regeneration. This value should be higher than RegenOff.

Description	Value
<b>Mnemonic (keyword):</b>	RegenOff
<b>CAN code:</b>	96
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RegenOn](#).

## RegenOn

RegenOn is the bus voltage in millivolts that activates the regeneration. This value should be lower than RegenOn.

Description	Value
<b>Mnemonic (keyword):</b>	RegenOn
<b>CAN code:</b>	95
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[RegenOff](#).

## RegenUsed

If RegenUsed = 0 the regeneration will not be activated and PC Suite will not display the regeneration configuration fields. This parameter will not change the other regeneration related parameters.

Description	Value
<b>Mnemonic (keyword):</b>	RegenUsed
<b>CAN code:</b>	378
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No for AG300/Yes for Central-i
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	1
<b>Implementation status:</b>	Firmware: 1.2.0

**Example:**

**See Also:**

## RelTrgt

RelTrgt is the relative target for point to point and for repetitive motions. RelTrgt defines the desired distance of motion from the current position. For point to point motion this will be the location in which the motion will end. For repetitive motions RelTrgt will determine one of the locations where the motor stops, and the current position will be the other location.

If RelTrgt is not 0 than AbsTrgt is ignored and RelTrgt is used to determine the actual target of the next motion.

RelTrgt can be changed during motion and the target of the current motion will change immediately.

Description	Value
<b>Mnemonic (keyword):</b>	RelTrgt
<b>CAN code:</b>	135
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

### Example:

If the current location is Pos = 1000, AbsTrgt = 5000 and RelTrgt = 0, the next point to point motion will end when Pos = 5000.

If the current location is Pos = 1000, AbsTrgt = 5000 and RelTrgt = 7000, the next point to point motion will end when Pos = 8000 (Using RelTrgt and adding it to the current location).

If the current location is Pos = 1000, AbsTrgt = 5000 and RelTrgt = 7000, the next repetitive motion will go to Pos = 8000 and then return to Pos = 1000 and so on.

### See Also:

[AbsTrgt](#)

## Reset

Reset will perform a software reset of the controller. All the processes will start the same as during power up. All the parameters that are saved in flash will be reloaded from flash and override their current values. Any parameters that are not saved to flash will have their default values.

Description	Value
<b>Mnemonic (keyword):</b>	Reset
<b>CAN code:</b>	234
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Return

Return will cause a jump back to the user program that will continue execution on the next line after a function call.

Note:

Use ProgHalt at the end of the program if your program is not an endless loop. Otherwise execution will continue into the first function and the “return” keyword will cause an error.

Description	Value
<b>Mnemonic (keyword):</b>	Return
<b>CAN code:</b>	432
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[ProgFunc](#), [ProgFuncCall](#)

## RevPLim

RevPLim is the reverse position limit in user units. This is the minimum allowed position for the motor. If the position exceeds this limit the motor is stopped. It will not be possible to start a new motion in the negative direction. Only positive motion is allowed until the position returns into the limits.

Description	Value
<b>Mnemonic (keyword):</b>	RevPLim
<b>CAN code:</b>	82
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	-2,000,000,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FWDPLim](#)



## RLType

Description	Value
<b>Mnemonic (keyword):</b>	RLType
<b>CAN code:</b>	375
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Rm

Description	Value
<b>Mnemonic (keyword):</b>	Rm
<b>CAN code:</b>	373
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	100,000
<b>Default value:</b>	1,000
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## RNDDebug

Description	Value
<b>Mnemonic (keyword):</b>	RNDDebug
<b>CAN code:</b>	1022
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	30
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## RptWait

RptWait – repeat wait in milliseconds. In repeat mode the motor moves back and forth between two points. Using RptWait the user can determine how long the motor will remain in each end point before it starts the new motion.

Since repeat mode is normally used for tuning, adding this waiting time is useful mainly for the following reasons:

It is easier to read recorded graphs if the end of one motion is visibly separate from the beginning of the next

The effects of decelerating in one motions and all their transients are allowed to decay before the next motion begins, and it is also easier to see them on a graph

If this type of motion continues for a long time, and high acceleration and decelerations are used, the user may want to add some wait time to prevent heating of system components

Description	Value
<b>Mnemonic (keyword):</b>	RptWait
<b>CAN code:</b>	147
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	131,071,993
<b>Default value:</b>	0
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[StopRep](#), [AbsTrgt](#), [RelTrgt](#), [MotionMode](#)

## RSBaud

RSBaud determines the baud rate for RS232. See the hardware manual to use the DIP switches to set the baud rate to default.

The value of RSBaud is checked during power up. To change the controller's baud rate set RSBaud to the desired value, then use "Save" to save the new baud rate to the flash. Cycle power for the new value to take effect.

The values of RSBaud are:

Value	Baud Rate
1	9600 bps
2	19200 bps
3	38400 bps
4	115200 bps

The default baud rate is 115200

Description	Value
<b>Mnemonic (keyword):</b>	RSBaud
<b>CAN code:</b>	79
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	No
<b>Min value:</b>	1
<b>Max value:</b>	4
<b>Default value:</b>	4
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Save

Save will save the parameters to a non-volatile (flash) memory.

After the Save command is entered, first the previous contents of the parameter area in the non-volatile memory is erased. Then, all the parameters that should be saved to flash are copied from the volatile memory to the flash.

To know if a parameter is saved to flash see the attribute table for the parameter in this document.

Description	Value
<b>Mnemonic (keyword):</b>	Save
<b>CAN code:</b>	232
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

Suppose that the current value of MaxSpeed is 0, because this is its default value. To be able to move the motor MaxSpeed must be changed.

The user enters MaxSpeed = 100000.

A query of MaxSpeed will return 100000 and motions can be performed.

However, if the power to the controller is turned off, on the next power up the value of MaxSpeed will be 0 again.

To save the value:

MaxSpeed = 100000

Save

On the next power up the saved value will be retrieved and used.

### See Also:

[Load](#)

## ScheduleGains

Description	Value
<b>Mnemonic (keyword):</b>	ScheduleGains
<b>CAN code:</b>	274
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ScheduleMode

Gain Scheduling is the process in which the controller automatically switches between different sets of control filter parameters as a function of some pre-defined criteria. It may be as a function of the motor speed (very low velocities do require a different control filter), or motion/settling/no motion, or distance to target and so on.

Agito controllers, at the moment, provide gain scheduling for the following control filter parameters: PosGain, VelGain, VelKi, VelFFW (not supported by all controllers) and AccFFW.

All these parameters are arrays, with a typical (product dependent) size of 5 elements. Therefore, up to 5 sets of control filters can be defined.

Set number [1] (PosGain[1], VelGain[1], VelKi[1], AccFFW[1]) is the default set that is used after power on or reset and if the gain scheduling is not active.

Relevant parameters:

- ScheduleSet:

Reports the number of the currently activated control set. It is "1" upon power on or reset and it is then controlled by the Gain Scheduling algorithm, according to its operation mode (see below: ScheduleMode).

ScheduleSet can be also set manually by the user, if the ScheduleMode is set to manual mode.

- ScheduleMode:

Defines the Gain Scheduling operation mode. Few modes are supported:

- **SCHEDULE\_MODE\_NONE** 0  
Gain scheduling is disabled.
- **SCHEDULE\_MODE\_MANUAL\_DINPORT** 1  
The active control set can be controlled manually (by writing to ScheduleSet), or it can be controlled using one of the digital inputs by proper setting of DinMode[ ].
- **SCHEDULE\_MODE\_OPTIMAL\_SETTLING\_BY\_TIME** 2  
Set 1 is used during motion, Set 2 is used during settling and Set 3 is used at all other times.  
Settling is a user defined time, in msec, that starts from end-of-motion, using the ScheduleTime parameter.
- **SCHEDULE\_MODE\_OPTIMAL\_SETTLING\_BY\_IN\_TARGET** 3  
Set 1 is used during motion, Set 2 is used during settling and Set 3 is used at all other times.  
Settling is the time from end-of-motion till the axis is settled into the target position (refer to InTargetStat, InTargetTime, InTargetTol).
- **SCHEDULE\_MODE\_BY\_VELOCITY\_RANGE** 4  
Set 1 is used while velocity is below ScheduleVel[1], Set 2 is used while velocity is below ScheduleVel[2] (and above ScheduleVel[1]) and so on ... Set 5 is used while velocity is above ScheduleVel[4].
- **SCHEDULE\_MODE\_BY\_POSITION\_RANGE** 5  
Set 1 is used while position is below SchedulePos[1],



Set 2 is used while position is below SchedulePos[2] (and above SchedulePos[1]) and so on ... Set 5 is used while position is above SchedulePos[4].

- **SCHEDULE\_MODE\_BY\_QUIET\_STANDING** 6  
Set 1 is used after a user defined time (using ScheduleTime parameter) of no-motion.  
Set 2 is used at all other times (during motions and for this defined time after the motion).
- **SCHEDULE\_MODE\_BY\_PD\_PULSES** 7  
Set 1 is used after a user defined time (using ScheduleTime parameter) of no input pulses at the pulse/direction input port.  
Set 2 is used at all other times (during incoming pulses and for this defined time after the last pulse).
- **SCHEDULE\_MODE\_BY\_TEMPERATURE\_RANGE** 8  
Supported only by some of Agito's controllers. Please check with Agito.  
Set 1 is used while the motor temperature (MotorTemp) is below ScheduleTemp[1],  
Set 2 is used while the temperature is below ScheduleTemp[2] (and above ScheduleTemp[1]) and so on ... Set 5 is used while the temperature is above ScheduleTemp[4].
- **SCHEDULE\_MODE\_BY\_VELOCITY\_RANGE\_INTERPOLATED** 9  
Like mode 4 above, but the gains are not changed instantly when passing from one velocity range to another, but instead changes gradually (linear interpolation) within each velocity range, between the values of the control sets defined for each range.

An interpolation between control set (i) and control set (i+1) is used in the range of velocity between ScheduleVel[i-1] and ScheduleVel[i], for i=1 to 4. ScheduleVel[0] is considered to be 0. Control set 5 is used while velocity is above ScheduleVel[4].

- **SCHEDULE\_MODE\_BY\_POSITION\_RANGE\_INTERPOLATED** 10  
Like mode 5 above, but the gains are not changed instantly when passing from one position range to another, but instead changes gradually (linear interpolation) within each position range, between the values of the control sets defined for each range.

An interpolation between control set (i) and control set (i+1) is used in the range of position between SchedulePos[i-1] and SchedulePos[i], for i=1 to 4. SchedulePos[0] is considered to be 0. Control set 5 is used while position is above SchedulePos[4].

- **SCHEDULE\_MODE\_FOR\_CNC\_MOTIONS** 11  
To be used for CNC motions, to improve the tracking performance during corners and following the corner.  
The 1st control set is used when not in CNC motion and when in motion, but in segments that are non-motion segments (like: Delay).

The 2nd control set is used during linear motion segments.

The 3rd control set is used during other types of motion segments (targeting corners).

The 4th control set is used for a given time (ScheduleTime) after the non-linear motion segment (targeting the time after exiting the corner segment)

- **ScheduleTime:**

Defines a period of time, in [msec], that is used by some of the gain scheduling modes (see above).

- **SchedulePos[]:**

Defines a set of position values that is used by the gain scheduling mode SCHEDULE\_MODE\_BY\_POSITION\_RANGE, as explained above.

- **ScheduleVel[]:**

Defines a set of velocity values that is used by the gain scheduling mode SCHEDULE\_MODE\_BY\_VELOCITY\_RANGE, as explained above.

- **ScheduleTemp[]:**

Supported only by some of Agito's controllers.

Defines a set of temperature values that is used by the gain scheduling mode SCHEDULE\_MODE\_BY\_TEMPERATURE\_RANGE, as explained above.

Description	Value
<b>Mnemonic (keyword):</b>	ScheduleMode
<b>CAN code:</b>	260
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	10
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**See Also:**

[ScheduleTime](#), [ScheduleVel](#), [SchedulePos](#), [ScheduleSet](#), [ScheduleTemp](#), [InTargetTol](#), [InTargetTime](#), [InTargetStat](#), [VelGain](#), [VelKi](#), [PosGain](#) and [AccFFW](#).

## SchedulePos

Please refer to the [ScheduleMode](#) keyword page for detailed description of the Gain Scheduling function and its parameters.

Description	Value
<b>Mnemonic (keyword):</b>	SchedulePos
<b>CAN code:</b>	264
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	2,147,483,647
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ScheduleTime](#), [ScheduleVel](#), [ScheduleMode](#), [ScheduleSet](#), [ScheduleTemp](#), [InTargetTol](#), [InTargetTime](#), [InTargetStat](#), [VelGain](#), [VelKi](#), [PosGain](#) and [AccFFW](#).

## ScheduleSet

Please refer to the [ScheduleMode](#) keyword page for detailed description of the Gain Scheduling function and its parameters.

Description	Value
<b>Mnemonic (keyword):</b>	ScheduleSet
<b>CAN code:</b>	261
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	5
<b>Default value:</b>	1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ScheduleMode](#)

## ScheduleTemp

Please refer to the [ScheduleMode](#) keyword page for detailed description of the Gain Scheduling function and its parameters.

Description	Value
<b>Mnemonic (keyword):</b>	ScheduleTemp
<b>CAN code:</b>	273
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-20
<b>Max value:</b>	120
<b>Default value:</b>	25
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ScheduleMode](#)

## ScheduleTime

Please refer to the [ScheduleMode](#) keyword page for detailed description of the Gain Scheduling function and its parameters.

Description	Value
<b>Mnemonic (keyword):</b>	ScheduleTime
<b>CAN code:</b>	262
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	9,999
<b>Default value:</b>	31
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ScheduleMode](#)

## ScheduleVel

Please refer to the [ScheduleMode](#) keyword page for detailed description of the Gain Scheduling function and its parameters.

Description	Value
<b>Mnemonic (keyword):</b>	ScheduleVel
<b>CAN code:</b>	263
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	60,000,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ScheduleMode](#)

## SetPDPos

Description	Value
<b>Mnemonic (keyword):</b>	SetPDPos
<b>CAN code:</b>	156
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## SetPosition

SetPosition is used to assign a new value to the main axis position reading (and position reference). SetPosition = 0 will change the position reading to become 0. Any value can be used. Other related internal variables are updated accordingly to allow a smooth change whether the motor is on or off.

SetPosition can't be used in the following conditions (the controller will return with a suitable error code):

1. While the motor is in motion.
2. If Error mapping is activated.
3. If Auto Gain (Auto Inertia) is activated.
4. If the SetPosition value is outside of the range defined by RevPLim to FwdPLim (software position limits).
5. If the motor is enabled and Input Shaping is activated.

Description	Value
<b>Mnemonic (keyword):</b>	SetPosition
<b>CAN code:</b>	154
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## ShapingDamp

Please refer to the [ShapingOn](#) keyword page for detailed description of the input shaping function and parameters.

Description	Value
<b>Mnemonic (keyword):</b>	ShapingDamp
<b>CAN code:</b>	153
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	65,535
<b>Default value:</b>	32,768
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ShapingOn](#),

## ShapingFreq

Please refer to the [ShapingOn](#) keyword page for detailed description of the input shaping function and parameters.

Description	Value
<b>Mnemonic (keyword):</b>	ShapingFreq
<b>CAN code:</b>	152
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	32,768,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ShapingOn](#),

## ShapingOn

---

“Shaping” refers to Input Shaping algorithm.

Input Shaping algorithm is an algorithm that is applied on the position command (PosRef) in order to modify it so that oscillations at the load will be canceled.

Important notes:

The Input Shaping algorithm can cancel (or reduce) oscillations only if they are created as a result of the motion itself.

The Input Shaping algorithm is effective to cancel oscillations that are not observed at the encoder feedback, just as it is effective to cancel observed oscillations.

The algorithm is performed on the position reference, so it is not a part of the closed loop and has no effect on its stability of performance (for example: disturbance rejection).

The Input Shaping algorithm, with its implementation at Agito’s controllers, can reject (or reduce) the oscillations of 1 or 2 frequencies.

The user needs to measure the frequency of the oscillation and its damping factor. These values are entered to the controller as parameters (see below). Once the Input Shaping is activated, the relevant oscillations shall be cancelled.

The algorithm is quite tolerant to inaccuracies of the specified frequency and damping. However, the more accurate the input data, the better will be the rejection of this oscillation.

Important note:

The usage of the Input Shaping algorithm modifies the shape of the position reference profile. However, not only the shape is modified, but also its timing. With Input Shaping activated (with a single frequency), the profile time is extended by 1 cycle time of the frequency to reject. This means that the Input Shaping is effective for systems with oscillations of more than one cycle (low damping factor values).

Relevant parameters:

**ShapingOn:**

A value of "0" disables the Input Shaping algorithm.

A value of "1" enables it.

**ShapingFreq[]:**

Defines the 1st (ShapingFreq[1]) frequency to reject and the 2nd (ShapingFreq[2]) frequency to reject.

Use a value of "0" at ShapingFreq[2] for rejection of a single frequency.

The value of ShapingFreq[] is the frequency value multiplied by 65536.

For example, to reject a frequency of 10.25Hz, use: ShapingFreq[1]= 671744.

This enables fractional frequencies.

**ShapingDamp[]:**

Defines the 1st (ShapingDamp[1]) damping factor and the 2nd (ShapingDamp[2]) damping factor.

The value of ShapingDamp[] is the damping factor value (a value between 0 to 1) multiplied by 65536.

For example, to define a damping factor of 0.1, use: ShapingDamp[1]= 6554.

Description	Value
<b>Mnemonic (keyword):</b>	ShapingOn
<b>CAN code:</b>	151
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ShapingFreq](#), [ShapingDamp](#).

## SinCosInPort

Description	Value
<b>Mnemonic (keyword):</b>	SinCosInPort
<b>CAN code:</b>	272
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 4
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## SinCosSetup

Description	Value
<b>Mnemonic (keyword):</b>	SinCosSetup
<b>CAN code:</b>	271
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 21
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Speed

The value of Speed determines the desired speed of the motion in UsrUnits/Sec. Speed can be changed on the fly during motion. If the current motion is in the acceleration phase or in the constant speed phase it will change to the updated speed.

Description	Value
<b>Mnemonic (keyword):</b>	Speed
<b>CAN code:</b>	138
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-60,000,000
<b>Max value:</b>	60,000,000
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

### Example:

Speed = 10,000

... (All other motion parameters and mode)

Begin

The above sequence starts a motion with desired speed of 10,000.

If after the actual velocity reaches 10,000 Speed is changed:

Speed = 20,000

(No Begin needed)

The motor will accelerate to 20,000 using the current set acceleration.

### See Also:

[Accel](#), [Decel](#), [AbsTrgt](#), [RelTrgt](#), [Begin](#)



## SpeedChgDir

Mode to switch automatically to new speed on Position.

0: If greater.

1: If smaller.

Description	Value
<b>Mnemonic (keyword):</b>	SpeedChgDir
<b>CAN code:</b>	347
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## SpeedChgNew

New value of speed.

Description	Value
<b>Mnemonic (keyword):</b>	SpeedChgNew
<b>CAN code:</b>	344
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-60,000,000
<b>Max value:</b>	60,000,000
<b>Default value:</b>	10,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[SpeedChgDir](#), [SpeedChgNew](#), [SpeedChgPos](#).

## SpeedChgOn

Mode to switch automatically to new speed on Position.

0: Disabled.

1: Enabled.

Description	Value
<b>Mnemonic (keyword):</b>	SpeedChgOn
<b>CAN code:</b>	345
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[SpeedChgNew](#), [SpeedChgPos](#), [SpeedChgDir](#).

## SpeedChgPos

Position to change speed.Change to SpeedChgNew.

Description	Value
<b>Mnemonic (keyword):</b>	SpeedChgPos
<b>CAN code:</b>	346
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	1,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[SpeedChgNew](#), [SpeedChgOn](#), [SpeedChgDir](#).

## SpringCurrFFW

Description	Value
<b>Mnemonic (keyword):</b>	SpringCurrFFW
<b>CAN code:</b>	596
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## SpringOn

Description	Value
Mnemonic (keyword):	SpringOn
CAN code:	592
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	Yes
Min value:	0
Max value:	1
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:

## SpringPHigh

Description	Value
<b>Mnemonic (keyword):</b>	SpringPHigh
<b>CAN code:</b>	594
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	10,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## SpringPLow

Description	Value
Mnemonic (keyword):	SpringPLow
CAN code:	593
Type:	Parameter
Access:	Read / Write
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	Yes
Axis related:	Yes
Min value:	-2,147,483,648
Max value:	2,147,483,647
Default value:	-10,000
User units:	In user units
Implementation status:	Implemented

Example:

See Also:



## SpringPosFFW

Description	Value
<b>Mnemonic (keyword):</b>	SpringPosFFW
<b>CAN code:</b>	595
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-100,000
<b>Max value:</b>	100,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## StatReg

StatReg is a status register. The bits of StatReg show the following statuses:

Bit	Status
0	Commutation done bit
1	Regeneration is on
2	Reserved
3	Maximum bus voltage exceeded (MaxVBus)
4	Under minimum bus voltage (MinVBus)
5	Main relay is on (for products with main relay)
6	Absolute maximum bus voltage exceeded (MaxVBusAbs)
8..7	<p>Bus voltage warning:</p> <p>00 – No warning. Bus voltage is deeply within its range.</p> <p>01 – Low warning. <math>V_{Bus} \geq (0.88 * MaxV_{Bus})</math>, or <math>V_{Bus} \leq (1.12 * MinV_{Bus})</math></p> <p>10 – Medium warning. <math>V_{Bus} \geq (0.92 * MaxV_{Bus})</math>, or <math>V_{Bus} \leq (1.08 * MinV_{Bus})</math></p> <p>11 – High warning. <math>V_{Bus} \geq (0.96 * MaxV_{Bus})</math>, or <math>V_{Bus} \leq (1.04 * MinV_{Bus})</math></p> <p>Bit 7 is the LSB.</p>
10..9	<p>Current reference warning:</p> <p>00 – No warning. Current reference (CurrRef) is deeply within its range.</p> <p>01 – Low warning. <math>Abs(CurrRef) &gt; (0.88 * PeakCL)</math></p> <p>10 – Medium warning. <math>Abs(CurrRef) &gt; (0.92 * PeakCL)</math></p> <p>11 – High warning. <math>Abs(CurrRef) &gt; (0.96 * PeakCL)</math></p> <p>Bit 9 is the LSB.</p>
12..11	<p>Power unit temperature warning:</p> <p>00 – No warning. PwrTemp is deeply within its range.</p> <p>01 – Low warning. <math>PwrTemp &gt; (0.88 * MaxPwrTemp)</math></p> <p>10 – Medium warning. <math>PwrTemp &gt; (0.92 * MaxPwrTemp)</math></p> <p>11 – High warning. <math>PwrTemp &gt; (0.96 * MaxPwrTemp)</math></p> <p>Bit 11 is the LSB.</p>

14..13	<p><b>General control saturation warning:</b></p> <p>This warning bits indicates how deep the control algorithms are in one of these saturations:</p> <p>Velocity saturation (MaxVel) or Current saturation (PeakCL or ContCL when continuous current limitation is active or limits by analog inputs when activated or by fixed values when activated) or voltage saturation (MaxPWM).</p> <p>In case that any of these saturations is activated (maximal or minimal values) at a given control cycle (sample), a counter is increased. If this counter, over a period of 1 second, is higher than the following defined thresholds, the proper warning state is signaled at this status bits.</p> <p>This warning state is updated once per second and stays unchanged for a period of 1 second (so the relevant LEDs at the PC Suite can be easily observed by the user).</p> <p>This warning is extremely important, as once the control algorithms enters one of its saturations, it means that the tracking performance are limited to some level (depend how deep is the saturation).</p> <p>The depth of the saturation is measured as the period of time at which at least one of the saturations was activated, within a period of 1 second. For example, a saturation depth of 100msec means that at least during an accumulated period of 100ms, within a period of 1 second, the control algorithms hit one of the saturations. This means: 10% of the control cycles were in saturation (one of them, not necessarily the same one at all cycles), or at least 1638 cycles (typically with Agito controllers the sampling frequency is 16384 Hz).</p> <p>Note that the warning here is signaled starting from very low saturation depths. This is because saturation is a critical state for motion performance and that sometimes the user is not aware to the saturation state. With these status bits, the event of saturation can be easily noticed.</p> <ul style="list-style-type: none"> <li>00 – No warning. The saturation depth is less than 0.2%.</li> <li>01 – Low warning. The saturation depth is more than 0.2%.</li> <li>10 – Medium warning. The saturation depth is more than 1%.</li> <li>11 – High warning. The saturation depth is more than 5%.</li> </ul> <p>Bit 13 is the LSB.</p> <p>Note that bits 21 to 23, as described below, provides the momentary status of each saturation (velocity, current and voltage).</p>
16..15	Motor temperature warning:

	<p>00 – No warning.  MotorTemp is deeply within its range.</p> <p>01 – Low warning.  MotorTemp &gt; (0.88 * MaxMotorTemp)</p> <p>10 – Medium warning.  MotorTemp &gt; (0.92 * MaxMotorTemp)</p> <p>11 – High warning.  MotorTemp &gt; (0.96 * MaxMotorTemp)</p> <p>Bit 15 is the LSB.</p> <p>(These bits are in use only from FW version 1.4.0)</p>
17	Reverse hardware limit is activated. This is identical to the status at LimitsStat.
18	Forward hardware limit is activated. This is identical to the status at LimitsStat.
19	The position reference (PosRef) is smaller than the software position forward limit (RevPLim).
20	The position reference (PosRef) is greater than the software position forward limit (FwdPLim).
21	<p>Current saturation is activated. The current saturation is activated if abs(CurrRef) is larger than one of: PeakCL or ContCL when continuous current limitation is active, or limits by analog inputs when activated, or by fixed values when activated.</p> <p>Note that in contrast to bits 14..13 that are described above, this bit reflects the momentary state of the saturation (as was activated or not activated during the recent control cycle).</p>
22	<p>Voltage saturation is activated. The voltage saturation refers to Va or Vb or Vc, as compared to <math>\pm</math>MaxPWM.</p> <p>Note that in contrast to bits 14..13 that are described above, this bit reflects the momentary state of the saturation (as was activated or not activated during the recent control cycle).</p>
23	<p>Velocity saturation is activated. The velocity saturation refers to VelRef, as compared to <math>\pm</math>MaxVel.</p> <p>Note that in contrast to bits 14..13 that are described above, this bit reflects the momentary state of the saturation (as was activated or not activated during the recent control cycle).</p>
25..24	<p>Other warning.  This is a general warning, for various purposes as follows:</p> <p>00 – No warning.</p> <p>01 – Low warning.  Currently not in use.</p> <p>10 – Medium warning.  I<sup>2</sup>t current limit is activated.</p> <p>11 – High warning.  Currently not in use.</p>

	Bit 24 is the LSB.
26	<p>The definition of at least one of the high order (bi-quad) filters, at the velocity control loop or at the position control loop, was modified by the user. This bit is set after one of the following parameters is assigned with a new value: VelFiltDef[], VelFiltOn[], PosFiltDef[] or PosFiltOn[].</p> <p>This bit reflects a non-valid state, at which the definitions of the filters are not correlated to the internal coefficients of the filters.</p> <p>Note that as a result, the controller will not allow to enable the axis (MotorOn=1) before CalcFilters is successfully executed.</p>
27	<p>CalcFilters execution was failed. This means that the internal coefficients of the bi-quad filters do not match the definition of the filters.</p> <p>Note that as a result, the controller will not allow to enable the axis (MotorOn=1) before CalcFilters is successfully executed.</p>
28	Dynamic brake status bit. It is set to "1" when the dynamic brake is activated.
29	Static brake lock request. It is set to "1" when the static brake is commanded to Lock state and to "0" when the static brake is (requested to) release.
30	Home switch is enabled. Valid only if a discrete input has been programmed as Home input for this axis.

The "Warning" bits, as described above, can be used by the user to identify an "almost" fault situation.

Note that "Agito PC Suite" software uses the bits values of this status parameter to control the LEDs at its status panel. The warning statuses, which have 4 values (none, low, medium and high) are reflected as a multi-color LED at the PC Suite status panel (off, yellow, orange and red, respectively).

#### See Also:

[MotionStat](#), [MotionReason](#), [ConFlt](#), [MotorOn](#), [VBus](#), [MaxVBus](#), [MinVBus](#), [CurrRef](#), [PeakCL](#), [PwrTemp](#), [MaxPwrTemp](#), [MaxVel](#), [ContCL](#), [CurrLimMode](#), [CurrLimFwd](#), [CurrLimRev](#), [LimitsStat](#), [RevPLim](#), [FwdPLim](#), [PosRef](#), [Va](#), [Vb](#), [Vc](#), [MaxPWM](#), [VelRef](#), [PeakTime](#), [VelFiltDef\[\]](#), [VelFiltOn\[\]](#), [PosFiltDef\[\]](#), [PosFiltOn\[\]](#) and [CalcFilters](#)

## StepBits

Description	Value
<b>Mnemonic (keyword):</b>	StepBits
<b>CAN code:</b>	256
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	2
<b>Max value:</b>	16
<b>Default value:</b>	2
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## StepInMotCurr

Description	Value
<b>Mnemonic (keyword):</b>	StepInMotCurr
<b>CAN code:</b>	255
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## StepInPosCurr

Description	Value
<b>Mnemonic (keyword):</b>	StepInPosCurr
<b>CAN code:</b>	254
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## Stop

The Stop command will cause the current motion to stop using the deceleration defined in Decel. If Stop is sent when no motion is in progress it has no effect.

Description	Value
<b>Mnemonic (keyword):</b>	Stop
<b>CAN code:</b>	132
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## StopCNCA

Description	Value
<b>Mnemonic (keyword):</b>	StopCNCA
<b>CAN code:</b>	456
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## StopECAM

StopECAM will stop the ECAM motion after the current cycle using the deceleration part of the table.

Description	Value
<b>Mnemonic (keyword):</b>	StopECAM
<b>CAN code:</b>	310
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[ECAMStart](#), [ECAMStartCyc](#), [ECAMEndCyc](#), [ECAMEnd](#), [ECAMCycles](#), [ECAMInterp](#), [ECAMTableNum](#), [ECAMGap](#), [ECAMMasterIni](#), [ECAMCycCount](#), [ECAMMaster](#)

## StopFIFO

Refer to the [FIFOType](#) keyword page for a full description of the FIFO motion mode and all related keywords.

Description	Value
<b>Mnemonic (keyword):</b>	StopFIFO
<b>CAN code:</b>	291
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[FIFOType](#)

## StopOnHome

Description	Value
<b>Mnemonic (keyword):</b>	StopOnHome
<b>CAN code:</b>	169
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## StopOnIndex

The StopOnIndex parameter defines if to stop the motion when an index (of the main encoder) is detected.

If StopOnIndex == 0, the motion will ignore index events.

If StopOnIndex == 1, the motion will stop when an index event is detected.

Important notes:

- StopOnIndex will stop the motion only in Jogging Motion Mode.
- The reason of the end-of-motion will appear at the MotionReason value (it will be 11). Please refer to the MotionReason keyword page.
- The index is detected by software. It is checked each sample of the controller (typically with Agito's controllers, the sampling rate is 16384 [Hz], meaning sample time of 61 [μsec]). Therefore, the jogging speed must be slow enough to ensure that the index is detected. Practically, set the jogging speed to any value below 8000 [counts/sec].
- Once the index is detected:
  - The index position is stored at the IndexPos parameter.
  - StopOnIndex is cleared to "0".
  - The motion starts to decelerate (using Decel parameter) till it stops.
- The motor will not stop on the index location till the deceleration process has some deceleration distance. The user can use the value at IndexPos to know the actual index position. The user can also set the deceleration value (Decel) to be very high comparing to the jogging speed, so that the deceleration distance will be as small as desired (even below 1 [count]).

Description	Value
<b>Mnemonic (keyword):</b>	StopOnIndex
<b>CAN code:</b>	167
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Homining](#), [MotionReason](#), [IndexStat](#), [IndexPos](#), [AuxIndexStat](#) and [AuxIndexPos](#).

## StopRep

StopRep stops a repetitive motion. If StopRep is entered during any other motion it is ignored. This is a special type of Stop. The motion will not start an immediate deceleration, but stop only after the current motion ends, at the beginning point of the motion. To stop the motion immediately, without waiting for the motor to return to the originating point of the motion use Stop.

Description	Value
<b>Mnemonic (keyword):</b>	StopRep
<b>CAN code:</b>	148
<b>Type:</b>	Command
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	0
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Stop](#), [RptWait](#)

## StopVec

Description	Value
Mnemonic (keyword):	StopVec
CAN code:	645
Type:	Command
Access:	Read only
Allowed in motion:	Yes
Allowed with motor on:	Yes
Save to flash:	No
Axis related:	Yes
Min value:	0
Max value:	0
Default value:	0
User units:	No user units
Implementation status:	Implemented

Example:

See Also:



## StuckCurr

StuckCurr, together with the other parameters listed below, defines the conditions in which the motor is considered stuck. If the motor is stuck it is disabled.

If the current applied to the motor exceeds StucCurr (mA) for a time longer than StuckTime, and the velocity is still below StuckVel, the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	StuckCurr
<b>CAN code:</b>	86
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[StuckTime](#), [StuckVel](#)

## StuckTime

StuckTime, together with the other parameters listed below, defines the conditions in which the motor is considered stuck. If the motor is stuck it is disabled.

If the current applied to the motor exceeds StucCurr for a time longer than StuckTime (sample times), and the velocity is still below StuckVel, the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	StuckTime
<b>CAN code:</b>	88
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	131,071,993
<b>Default value:</b>	250
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[StuckCurr](#), [StuckVel](#)

## StuckVel

StuckVel, together with the other parameters listed below, defines the conditions in which the motor is considered stuck. If the motor is stuck it is disabled.

If the current applied to the motor exceeds StucCurr for a time longer than StuckTime, and the velocity is still below StuckVel (in UsrUnits/sec), the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	StuckVel
<b>CAN code:</b>	87
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	40,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[StuckCurr](#), [StuckTime](#)

## Targets

Description	Value
<b>Mnemonic (keyword):</b>	Targets
<b>CAN code:</b>	376
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 3
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Time

Time is a read only parameter that holds the time in seconds since power on.

Description	Value
<b>Mnemonic (keyword):</b>	Time
<b>CAN code:</b>	41
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## TorqCompFix

Refer the detailed description of the torque compensation function at the TorqCompMode keyword page.

Description	Value
<b>Mnemonic (keyword):</b>	TorqCompFix
<b>CAN code:</b>	390
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-5,000
<b>Max value:</b>	5,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[TorqCompMode](#),

## TorqCompMode

Torque compensation is an external addition to the torque command (CurrRef – Current command).

Two options are supported: Using analog input (with scaling) or using preset values.

As analog input, it uses AInPort[4]. This analog input (as all the analog inputs) is processed using: offset, gain, filter and dead band, so it can be properly adjusted for the application needs. The relevant parameters are:

AInOffset[4]  
AInGain[4]  
AInFilt[4]  
AInDB[4]

As preset values, the torque compensation uses the array parameter: TorqCompFix[]. It has the size of 5 and so the user can define 5 different fixed compensation values.

The TorqCompMode defines the behavior of the torque compensation, as follows:

- If TorqCompMode == -1 (default value): No compensation
- If TorqCompMode == 0 Use analog input:  
CurrRef = CurrRef + AInPort[4]
- If TorqCompMode is between 1 to 5: Use fixed value  
CurrRef = CurrRef + TorqCompFix[TorqCompMode]

The torque compensation is relevant for Position and Velocity modes only.

Description	Value
<b>Mnemonic (keyword):</b>	TorqCompMode
<b>CAN code:</b>	391
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-1
<b>Max value:</b>	5
<b>Default value:</b>	-1
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[TorqCompFix\[\]](#),

## UserMode

UserMode is a parameter used to activate special algorithms within the controller. These algorithms are for the most part custom made for dedicated purposes.

By assigning a very specific value to UserMode, the user can activate a pre-defined algorithm that was included in the controller software for this user needs.

Relevant customers will be informed about possible and relevant values of UserMode to activate their special algorithms.

All other customers are advised to keep this parameter as 0, to avoid accidental operation of a non-documented special and application dedicated feature.

Description	Value
<b>Mnemonic (keyword):</b>	UserMode
<b>CAN code:</b>	77
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## UserParam

Description	Value
<b>Mnemonic (keyword):</b>	UserParam
<b>CAN code:</b>	624
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 50
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## UserPWM

There are two parameters used to set the user PWM module, UserPWM and UserPWMDiv.

UserPWM controls the duty cycle of each user PWM output. This is the compare value such that when the internal counter < UserPWM[N] the output value is 1 else the output value is 0. This allows us to generate a PWM signal where we can vary the duty cycle with resolution of 1/4096(12bit).

There are two UserPWM—UserPWM1 and UserPWM2. UserPWM1 and UserPWM2 have the same PWM frequency and are fully synchronized. Yet, the duty cycle of each one of them is set separately.

UserPWMDiv controls the clock frequency for the user PWM module. The value of UserPWMDiv is between 0 to 15, where 0 makes the user PWM clock count at 40 MHz, 1 will make it 20 MHz, 2 will make it 10 MHz etc. It counts 4096 counts (we will call this internal counter) before going back to 0.

$$\text{The PWM clock Frequency} = 80\text{MHz} / 2^{\text{(UserPWMDiv + 1)}}$$

And the frequency of the PWM signal would be:

$$\text{PWM Frequency} = \text{PWM clock frequency} / 4096$$

After the setting of these two parameters, users can use the discrete outputs selector to output the PWM signal to any output. Please refer to [DOutSelect](#) manual page for full description.

### Note:

UserPWM is not supported for now at Central-I systems but it can be easily added upon request.

UserPWM is only available from v1.3.0 and above of the firmware with FPGA version v1.3.0 and above.

Description	Value
<b>Mnemonic (keyword):</b>	UserPWM
<b>CAN code:</b>	626
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	4095
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented(Valid from FW1.3.0 with AG300)

**Examples:**

AUserPWM[1]=1000; Then it will generate a PWM with duty cycle of  $1000/4096=24.4\%$ .

AUserPWMDiv=3; Then PWM clock Frequency= $80/2^{\textcolor{blue}{3} + 1}$  MHz=5 MHz and the resulted PWM frequency will be:

$$5\text{MHz} / 4096 = 1.22 \text{ KHz}$$

With these settings, the selector output would send out 1.22 KHz PWM output with 24.4%duty cycle.

**See Also:**

[UserPWMDiv](#), [DOutSelect](#).

## UserPWMDiv

Please refer to the UserPWM manual page for full description of the UserPWM feature.

Description	Value
<b>Mnemonic (keyword):</b>	UserPWMDiv
<b>CAN code:</b>	627
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	15
<b>Default value:</b>	9
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

AUserPWMDiv=3; Then PWM clock Frequency= $80/2^3 + 1$  MHz=5 MHz and the resulted PWM frequency will be:

$$5\text{MHz} / 4096 = 1.22 \text{ KHz.}$$

### See Also:

[UserPWM](#), [DoutSelect](#).

## UsrUnits

UsrUnits allows the user to read the position and its derivatives in units other than encoder counts. UsrUnits is the ratio between the desired unit and the encoder counts. A full explanation of how user units work can be found in the beginning of this document.

Example:

If the user wants to see the position reading in mm, and every 5 encoder counts are equivalent to 1mm, set UsrUnits to 5.

The position reading will now be received in mm, velocity in mm/sec and acceleration in mm/sec<sup>2</sup>.

Description	Value
<b>Mnemonic (keyword):</b>	UsrUnits
<b>CAN code:</b>	64
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	1
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	65,536
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

Example:

See Also:

[AuxUsrUnits](#), [PDUsrUnits](#)

## Va

Va returns the value of the voltage applied to phase “A” in percent of the full PWM. Phase “A” is as defined in the hardware reference guide.

Description	Value
<b>Mnemonic (keyword):</b>	Va
<b>CAN code:</b>	13
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Vb

Vb returns the value of the voltage applied to phase “B” in percent of the full PWM. Phase “B” is as defined in the hardware reference guide.

Description	Value
<b>Mnemonic (keyword):</b>	Vb
<b>CAN code:</b>	14
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VBus

VBus returns the reading of the bus voltage in Millivolts.

Description	Value
<b>Mnemonic (keyword):</b>	VBus
<b>CAN code:</b>	36
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

If the bus voltage is 150V, VBus returns 150,000

**See also:**



## Vc

Vc returns the value of the voltage applied to phase “C” in percent of the full PWM. Phase “C” is as defined in the hardware reference guide.

Description	Value
<b>Mnemonic (keyword):</b>	Vc
<b>CAN code:</b>	15
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## Vd

The Vd parameter is used only when Vector Control is active (refer to ControlMode keyword).

The Vd is the output of the PI controller at the Id current control loop.

Vd and Vq are used to calculate Va and Vb (the desired motor phase voltages).

Vd is in internal units (typically at Agito's controllers: value of 4577 at Vd refers to 100% PWM).

Vd is equal to zero when Vector Control is not active.

Description	Value
<b>Mnemonic (keyword):</b>	Vd
<b>CAN code:</b>	16
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VecAbsTrgt

The VecAbsTrgt is a status parameter that reports the target distance (from start of motion to its end) of the vector motion along the vector path. VecAbsTrgt is always positive.

The VecAbsTrgt is not used to define the vector motion. This is done using the RelTrgt or AbsTrgt of the member axes.

Description	Value
<b>Mnemonic (keyword):</b>	VecAbsTrgt
<b>CAN code:</b>	642
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAbsTrgt](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#),

## VecAccel

Description	Value
<b>Mnemonic (keyword):</b>	VecAccel
<b>CAN code:</b>	636
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#),

## VecArcCenter

For ARC vector, defines the location of the arc center, so the controller can calculate the radius. The VecArcCenter, like all other new keywords for the vector motion, is per axis. So, the coordinate of the arc center are defined by the VecArcCenter of the two member axes. Saved to Flash. Can't be modified while in motion.

Description	Value
<b>Mnemonic (keyword):</b>	VecArcCenter
<b>CAN code:</b>	633
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcDir](#), [VecEncRatio](#),

## VecArcDir

For ARC vector, defines the direction of the ARC. "0" for CCW, "1" for CW.

The VecArcDir, like all other new keywords for the vector motion, is per axis. However, only the VecArcCenter of the axis which was used for the Begin command will be used.

2 axes are defined for an ARC motion. The ARC is performed at the plain of these two axes, with the third axis not moving.

The two axes are defined with an important order, meaning, for example: B, C is not like C, B.

The first axis is the X axis. The second is the Y axis. Then CCW motion is around the "Z" axis, where X axis is moving toward Y axis.

Note: need to check that this definition is aligned with the CNC definition of ARC.

Saved to Flash. Can't be modified while in motion.

Description	Value
<b>Mnemonic (keyword):</b>	VecArcDir
<b>CAN code:</b>	634
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcCenter](#), [VecEncRatio](#),

## VecDecel

Description	Value
<b>Mnemonic (keyword):</b>	VecDecel
<b>CAN code:</b>	637
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#),

## VecdPosRef

A status parameter that reports the derivative of the vector motion position reference (derivative of VecPosRef). VecdPosRef is always positive.

Description	Value
<b>Mnemonic (keyword):</b>	VecdPosRef
<b>CAN code:</b>	644
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## VecEmrgDec

Description	Value
<b>Mnemonic (keyword):</b>	VecEmrgDec
<b>CAN code:</b>	638
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	100
<b>Max value:</b>	2,000,000,000
<b>Default value:</b>	100,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VecEncRatio

Used to compensate for different encoder resolutions. Each axis is scaled according to its resolution.

The actual ratio inside the controller is scaled by /256. So VecEncRatio=256 means a ratio of 1. A value of 260 means ratio of 260/256.

Details about how to use the EncRatio: TBD (during implementation).

Implementation must be done in a way not to cause accumulated position errors and final target position must be reached accurately.

Saved to Flash. Can't be modified while in motion. Range 256 (ratio of 1) to 25600 (ratio of 100).

Description	Value
<b>Mnemonic (keyword):</b>	VecEncRatio
<b>CAN code:</b>	632
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	256
<b>Max value:</b>	25,600
<b>Default value:</b>	256
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#),

## VecJerk

Description	Value
<b>Mnemonic (keyword):</b>	VecJerk
<b>CAN code:</b>	639
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	9
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VecMemberAxes

Used (bitwise) to define the axes participating in the vector motion.

Saved to Flash. Can't be modified while in motion.

Description	Value
<b>Mnemonic (keyword):</b>	VecMemberAxes
<b>CAN code:</b>	631
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	255
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VecMotionStat

Description	Value
<b>Mnemonic (keyword):</b>	VecMotionStat
<b>CAN code:</b>	641
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VecPause

A value of "1" pauses the vector motion by setting the speed to 0 (motion is decelerated till stopping). A value of "0" continue the motion normally (if paused before, it will accelerate to the VecSpeed again).

Not Saved to Flash. At power-up gets its default value: "0".

Description	Value
<b>Mnemonic (keyword):</b>	VecPause
<b>CAN code:</b>	640
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VecPosRef

The VecPosRef is a status parameter that reports the current position reference of the vector motion profile (along the vector path). It starts from a value of 0 and upon end of motion, reaches the value of VecAbsTrgt.

VecPosRef is always positive.

Description	Value
<b>Mnemonic (keyword):</b>	VecPosRef
<b>CAN code:</b>	643
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#), [VecSpeed](#).

## VecSpeed

Description	Value
<b>Mnemonic (keyword):</b>	VecSpeed
<b>CAN code:</b>	635
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	60,000,000
<b>Default value:</b>	10,000
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#),



## VecType

Defines if the requested vector motion is linear (VecType = 0) or ARC (VecType = 1).

Saved to Flash. Can't be modified while in motion.

Near future need: VecType = 2 for combined ARC (main motion) and linear (other axes).

Description	Value
<b>Mnemonic (keyword):</b>	VecType
<b>CAN code:</b>	630
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VecAccel](#), [VecAbsTrgt](#), [VecArcCenter](#), [VecArcDir](#), [VecEncRatio](#), [VecSpeed](#).

## Vel

Vel[] is an array that reports the main encoder velocity in three different formats:

Vel[1] is the filtered value of the velocity. The factors of the filter are determined according to VelFilt.

Vel[2] is the raw velocity reading.

Vel[3] is the average speed over 16 samples

All the above are in UsrUnits / Sec

### Why is the velocity filtered?

The raw velocity reading is the derivative of the position. The position is an integer value that represents the number of encoder counts that were read. It is possible, and often happens, that the required velocity corresponds to a non-integer number of encoder pulses.

If for example, the actual velocity is 10.5 counts per sample time, the position after 1 sample time will be:

$Pos_1 = 10$  (since 0.5 count cannot be detected)

The calculated velocity for 16384 samples per second is  $10 * 16384 = 163840$  counts/sec

During the next sample time the motor passes 10.5 more counts. Since it started from an actual position of 10.5 it will reach position 21.

$Pos_2 = 21$

The controller calculates the position difference to detect the velocity. The calculated velocity is 11 counts per sample time, or  $11 * 16384 = 180224$

We receive a difference of 18384 between the velocity readings of two sample times even though in reality the velocity was constant.

Filtering the velocity has two benefits:

1. The user can see a smoothed value when looking at a graph and get a better indication of the actual average velocity
2. The filtered value is also used by the control to eliminate unnecessary high frequency noises.

The moving average is only for display purposes for the user. It is not used in the control.

Description	Value
Mnemonic (keyword):	Vel
CAN code:	5
Type:	Parameter
Access:	Read only
Allowed in motion:	Yes
Allowed with motor on:	Yes
Array with index range of:	1 : 4
Save to flash:	No
Axis related:	Yes
Min value:	-2,147,483,648
Max value:	2,147,483,647
Default value:	0
User units:	In user units
Implementation status:	Implemented

Example:

**See Also:**

[MaxVelErr](#), [VelRef](#), [VelErr](#).

## VelErr

VelErr returns the value of the velocity error in user units (UsrUnits)/Sec. The velocity error is the difference between the velocity command (VelRef) and the actual velocity (Vel). If the value of VelErr exceeds the maximum allowed position error as defined in MaxVelErr the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	VelErr
<b>CAN code:</b>	19
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

### Example:

If VelRef = 10000 and Vel = 9900 then VelErr = 100

### See Also:

[UsrUnits](#), [MaxVelErr](#), [VelRef](#), [Vel](#).

## VelFFW

The VelFFW is used to inject a value into CurrRef that is proportional to the profiler velocity (first derivative of the position reference: PosRef).

The relevant equation is:

$$\text{CurrRef}_k = (\text{Velocity Control Output})_k + \text{AccFFW} * (\text{PosRef}_k - 2 * \text{PosRef}_{k-1} + \text{PosRef}_{k-2}) / (256 * \text{Ts}) + \text{VelFFW} * (\text{PosRef}_k - \text{PosRef}_{k-1}) / (65536 * \text{Ts})$$

Where Ts is the sampling time of the controller, typically 61μs with Agito's controllers.

Note that for products (or operational modes) without current control loop (like for VCM motors), the PWM command actually equals to the CurrRef:  $V_a = \text{CurrRef}$ . In this case, VelFFW injects PWM command (motor phase voltage) that is proportional to the profiler velocity.

Description	Value
<b>Mnemonic (keyword):</b>	VelFFW
<b>CAN code:</b>	108
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	50,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[AccFFW](#), [PosRef](#), [PosGain](#), [VelGain](#), [VelKi](#), [CurrRef,Va](#) and [Vb](#).

## VelFiltDef

Please refer to the [VelFiltOn](#) keyword page for detailed description of the velocity bi-quad filters function.

Description	Value
<b>Mnemonic (keyword):</b>	VelFiltDef
<b>CAN code:</b>	121
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 10
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

## VelFiltOn

Agito's controllers include some bi-quad filters as part of the closed loop control filters. At the moment, a single bi-quad filter is supported at the position control (refer to the PosFiltOn[] keyword page) and two bi-quad filters are supported at the velocity control.

The first bi-quad filter at the velocity control loop is used to filter the velocity reading:  
 $Vel[1] = BiQuad\ Filter(Vel[2])$

Where  $Vel[2]$  is the derivative of the position reading.

This bi quad filter (Velocity 1st bi-quad filter) is defined using VelFiltOn[1], VelFiltDef[1] to VelFiltDef[5] and its coefficients are stored at VelFilt[1-5], all as explained below.

Typically, a second order low pass filter is used for this filter.

The second velocity bi-quad filter is a filter on the CurrRef command (the command to the current control loop, which is the output of the velocity control loop).

This bi quad filter (Velocity 2nd bi-quad filter) is defined using VelFiltOn[2], VelFiltDef[6] to VelFiltDef[10] and its coefficients are stored at VelFilt[6-10], all as explained below.

Typically, a second order low pass filter is used for this filter, or a notch filter, if required.

Filters are defined using the VelFiltOn[] and VelFiltDef[] parameters:

VelFiltOn[N] enables ("1") or disables ("0") the filter number N. When the filter is disabled, it actually means that the filter is equal to 1.

At the moment, as explained above, only VelFiltOn[1] and VelFilt[2] are used, to enable/disable the first and the second velocity advanced bi-quad filters.

VelFiltDef[] is used to define the filter type and its parameters:

VelFiltDef[] is divided into groups of 5 elements (currently only 2 groups are used). This means: VelFiltDef[1] to VelFiltDef[5] refer to the first filter. VelFiltDef[6-10] refer to the second filter and so on.

The first element of each group defines the filter's type.

The rest 4 elements define the filter characteristics (frequency, damping, ...) as relevant for each type of filter (not all of them use all the 4 elements).

The list of supported filter's types and their related parameters are described below.

Upon completing the definitions of VelFiltOn[] and VelFiltDef[] (and actually also any change of VelFiltOn[] or VelFiltDef[]), the user must send the CalcFilters command, so that this input data will be used to calculate the actual filter coefficients. These coefficients are stored at the VelFilt[] array (5 elements per each filter).

The VelFilt[] is a read only array.

The structure of the internal implementation of the filter is as follows (for the first filter, that is stored at VelFilt[1-5]):

$$YK = (XK * VelFilt[1] + XK-1 * VelFilt[2] + XK-2 * VelFilt[3] - YK-1 * VelFilt[4] - YK-2 * VelFilt[5]) / 65536$$

Where X is the input to the filter and Y is its output.

Notes:

Upon any change of VelFiltOn[] and VelFiltDef[], the controller will not allow enabling of the axis until a successful CalcFilters operation. This is to ensure that the control loop is using the filters defined by the user (at VelFiltOn[], VelFiltDef[]).

List of supported filters and their characteristic parameters (as defined within VelFiltDef[]):  
In the description below, the numeric value that represents each type of filter is the value to put at VelFiltDef[1] (the first element of the group of 5) or VelFiltDef[6] (for the second filter) and par1 to par4 are actually the values at VelFiltDef[2-5] (or [7-10] for the second filter).

```
//  
// Note: All frequency parameters (poles, zeros ...) that are described below are //  
// actually provided as Hz*100, not Hz. This is to enable frequencies with  
// fractions.  
//  
// 1 - LPF1: 1st order Low Pass Filter the par1 is the pole frequency in  
// Hz, par2, 3 and 4 aren't applicable.  
//  
// 2 - LPF2: 2nd order Low Pass Filter, par1 is the pole frequency in  
// Hz, par2 is the pole damping ratio. Par3 and 4 aren't applicable.  
//  
// 3 - LPF3: 2nd order Low Pass Filter with zero,  
// par1 is the pole frequency in  
// Hz, par2 is the pole damping ratio and par3 is the zero frequency  
// in Hz. par4 isn't applicable.  
//  
// 4 - LDLG2: 2nd order Lead/Lag filter  
// par1 is the lead first zero frequency, par2  
// is the lead second zero, par3 is the leg first pole and  
// par4 is the leg second pole.  
//  
// 5 - LDLG1: 1st order Lead/Lag filter  
// par1 is the lead zero frequency and par2 is the leg pole.  
//  
// 6 - LDLG1FP: 1st order Lead/Lag filter defined by the frequency at the  
// phase peak/min and the phase level.  
// par1 is the frequency at the phase's peak/min in Hz,  
// par2 in the required phase peak in degrees.  
//  
// 7 - LDLG2FP: 2nd order Lead/Lag filter defined by the frequencies at  
// the phases peak/min and the phases level (Approximately)  
// par1 is the 1st frequency at the phase peak/min in Hz,  
// par2 in the required phase peak in degrees at the 1st frequency.  
// par3 is the 2nd frequency at the phase peak/min in Hz,  
// par4 in the required phase in degrees at the 2nd frequency.  
//  
// 8 - NOTCH: Notch filter  
// par1 is the notch frequency in Hz, par2 is  
// the notch depth in dB and par 3 is the Notch width in Hz.  
// par4 isn't applicable.
```



```
//
// 9 –CLDLG: Complex Lead/Leg filter.
// par1 - frequency of the complex zero in Hz, par2 damping
// ratio of the complex zero, par3 is the frequency of the
// pole in Hz and par4 is the damping ratio of pole.
//
```

Description	Value
<b>Mnemonic (keyword):</b>	VelFiltOn
<b>CAN code:</b>	122
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	No
<b>Allowed with motor on:</b>	No
<b>Array with index range of:</b>	1 : 2
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[VelFiltDef\[\]](#), [VelFiltDef\[\]](#), [PosFiltOn\[\]](#), [PosFiltDef\[\]](#), [PosFiltOn\[\]](#) and [CalcFilters](#).

## VelGain

Agito controllers implement a position over velocity control filter scheme.

The velocity control filter includes a PI filter (VelGain and VelKi) and several bi-quad filters. The actual number of the filters may vary between the products.

VelGain[] is an array, to serve the gain scheduling mechanism. At any time, a given element of VelGain[] is used for the control filter calculations, based on the current decision of the gain scheduling mechanism.

The default element is VelGain[1].

The user may set all elements of VelGain[] to be equal, so the same gain value will be used with any decision of the gain scheduling mechanism. The user can set different values, so different gains will be used in different cases.

The relevant equations in the controller are (assuming no velocity bi-quad filter is used):

$VelErr = VelRef - Vel[1]$ . (note: refer to PosRef and Vel documentation)

for better understanding of this equation)

$Temp = VelErr * VelGain[As\ Selected\ By\ Scheduling]$

$VelIntegral = VelIntegral + Temp * VelKi[As\ Selected\ By\ Scheduling] * SAMPLE\_TIME$

$VelPIOutput = (Temp + VelIntegral) * 65536$

$CurrRef = VelPIOutput + \dots AccFFW$  (see AccFFW documentation).

It is important to note that following these equations, CurrRef itself is limited by the  $\pm PeakCL$  (PeakCL is a controller parameter).

Description	Value
<b>Mnemonic (keyword):</b>	VelGain
<b>CAN code:</b>	102
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	1,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosGain](#), [VelKi](#), [AccFFW](#), [PosFiltDef](#), [VelFiltDef](#), [PeakCL](#), [ScheduleMode](#).

## VelKi

Agito controllers implement a position over velocity control filter scheme.

The velocity control filter includes a PI filter (VelGain and VelKi) and several bi-quad filters. The number of the filters depends on the product.

VelKi[] is an array, to serve the gain scheduling mechanism. At any time, a given element of VelKi[] is used for the control filter calculations, based on the current decision of the gain scheduling mechanism.

The default element is VelKi[1].

The user may set all elements of VelKi[] to be equal, so the same gain value will be used with any decision of the gain scheduling mechanism, or it can set different values, so different gains will be used for each case.

The relevant equations in the controller are (assuming no velocity bi-quad filter is used):

$VelErr = VelRef - Vel[1]$ . (note: refer to PosRef and Vel documentation)

for better understanding of this equation)

$Temp = VelErr * VelGain[As\ Selected\ By\ Scheduling]$

$VelIntegral = VelIntegral + Temp * VelKi[As\ Selected\ By\ Scheduling] * SAMPLE\_TIME$

$VelPIOutput = (Temp + VelIntegral) * 65536$

$CurrRef = VelPIOutput + \dots AccFFW$  (see under AccFFW documentation).

It is important to note that following these equations, CurrRef itself is limited by the  $\pm PeakCL$  (PeakCL is a controller parameter).

Description	Value
<b>Mnemonic (keyword):</b>	VelKi
<b>CAN code:</b>	103
<b>Type:</b>	Parameter
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 5
<b>Save to flash:</b>	Yes
<b>Axis related:</b>	Yes
<b>Min value:</b>	0
<b>Max value:</b>	100,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[PosGain](#), [VelGain](#), [AccFFW](#), [PosFiltDef](#), [VelFiltDef](#), [PeakCL](#), [ScheduleMode](#).

## VelRef

VelRef returns the velocity reference generated by the internal profiler in user units (UsrUnits)/Sec.

Description	Value
<b>Mnemonic (keyword):</b>	VelRef
<b>CAN code:</b>	25
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-60,000,000
<b>Max value:</b>	60,000,000
<b>Default value:</b>	0
<b>User units:</b>	In user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[UsrUnits](#)

## VLogic

VLogic returns the current reading of the logic supply voltage. The nominal supply voltage is 5V. If the actual supply voltage is above or below the designated limits (hard coded) the motor is disabled.

Description	Value
<b>Mnemonic (keyword):</b>	VLogic
<b>CAN code:</b>	37
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

Normally VLogic = 5000

**See Also:**

## Vq

The Vq parameter is used only when Vector Control is active (refer to ControlMode keyword).

The Vq is the output of the PI controller at the Iq current control loop.

Vd and Vq are used to calculate Va and Vb (the desired motor phase voltages).

Vq is in internal units (typically at Agito's controllers: value of 4577 at Vq refers to 100% PWM).

Vq is equal to zero when Vector Control is not active.

Description	Value
<b>Mnemonic (keyword):</b>	Vq
<b>CAN code:</b>	17
<b>Type:</b>	Parameter
<b>Access:</b>	Read only
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	dynamic (depends on remote unit)
<b>Max value:</b>	dynamic (depends on remote unit)
<b>Default value:</b>	dynamic (depends on remote unit)
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**

[Vd](#), [Iq](#),

## WaitStatus

WaitStatus is a user program low level language keyword. The syntax related to “Compare” is usually generated automatically by the PC suite during compilation.

WaitStatus[Status type], status value

Waits until the status that is defined reaches the required value. The user program will continue only after the value is reached.

The types for StatusType:

Type	Status type
1	COUNTERDOWN1: Wait until CounterDown[1] reaches the value
2	COUNTERDOWN2: Wait until CounterDown[2] reaches the value
3	COUNTERDOWN3: Wait until CounterDown[3] reaches the value
4	COUNTERDOWN4: Wait until CounterDown[4] reaches the value
5	COUNTERUP1: Wait until CounterUp[1] reaches the value
6	COUNTERUP2: Wait until CounterUp[2] reaches the value
7	IN_MOTION: Wait until MotionStat in motion bit reaches the value
8	IN_RPT_WAIT: Wait until MotionStat in repeat wait bit reaches the value
9	IN_RPT_STOP: Wait until MotionStat stop bit reaches the value
10	IN_STOP_REQUEST: Wait until MotionStat stop request bit reaches the value
11	IN_ACCELERATION: Wait until MotionStat in acceleration bit reaches the value
12	IN_DECELERATION: Wait until MotionStat in deceleration bit reaches the value
13	IN_WAIT_END_SMOOTH: Wait until MotionStat smoothing ended bit reaches the value
14	IN_ECAM_STOP: Wait until MotionStat in ECAM stop bit reaches the value
15	IN_FIFO_STOP: Wait until MotionStat in FIFO stop bit reaches the value
16	COMMUTATION: Wait until StatReg commutation bit reaches the value
17	IN_TARGET: Wait until StatReg in target bit reaches the value
18	REC_TRIG_DETECTED_STATUS: Wait until RecStat trigger detected reaches the value
19	REC_COMPLETE_STATUS: Wait until RecStat recording complete reaches the value
20	DIN1: Wait until digital input 1 reaches the value
21	DIN2: Wait until digital input 2 reaches the value
22	DIN3: Wait until digital input 3 reaches the value
23	DIN4: Wait until digital input 4 reaches the value
24	DIN5: Wait until digital input 5 reaches the value
25	DIN6: Wait until digital input 6 reaches the value
26	DIN7: Wait until digital input 7 reaches the value
27	DIN8: Wait until digital input 8 reaches the value
28	DIN9: Wait until digital input 9 reaches the value
29	DIN10: Wait until digital input 10 reaches the value
30	DIN11: Wait until digital input 11 reaches the value
31	DIN12: Wait until digital input 12 reaches the value
32	DIN13: Wait until digital input 13 reaches the value
33	DIN14: Wait until digital input 14 reaches the value
34	MOTOR_ON_INPUT_STATUS: Wait until the input designated as motor on reaches the value
35	VEL_GAIN_CHANGE_STATUS: Wait until the input designated as gain change reaches the value
36	CLEAR_ABS_ENC_STATUS: Wait until the input designated as clear absolute encoder on reaches the value
37	CLEAR_IN_PULSES_STATUS: Wait until the input designated clear input pulses on reaches the value
38	REV_LIMIT_STATUS: Wait until the input designated as reverse limit reaches the value
39	FWD_LIMIT_STATUS: Wait until the input designated as forward limit reaches the value

40	TORQUE_LIMIT_ON_STATUS: Wait until the input designated as torque limit reaches the value
41	ALARM_RESET_STATUS: Wait until the input designated as alarm reset reaches the value
42	ABORT_INPUT_STATUS: Wait until the input designated as abort reaches the value
43	MODE_SWITCH_VEL_POS_STATUS: Wait until the input designated as vel / pos switch reaches the value
44	MODE_SWITCH_VEL_CUR_STATUS: Wait until the input designated as vel/cur switch reaches the value
45	MODE_SWITCH_POS_CUR_STATUS: Wait until the input designated as pos/cur switch reaches the value
46	ADD_FILTER_STATUS: Wait until the input designated as add filter reaches the value

Description	Value
<b>Mnemonic (keyword):</b>	WaitStatus
<b>CAN code:</b>	194
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Array with index range of:</b>	1 : 33
<b>Save to flash:</b>	No
<b>Axis related:</b>	Yes
<b>Min value:</b>	-2,147,483,648
<b>Max value:</b>	2,147,483,647
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

**Example:**

**See Also:**



## WaitTime

WaitTime is the time in milliseconds that the user program should wait before continuing to the next line.

The format of this keyword is:

WaitTime, milliseconds

This command is only allowed in a user program.

Description	Value
<b>Mnemonic (keyword):</b>	WaitTime
<b>CAN code:</b>	193
<b>Type:</b>	Command
<b>Access:</b>	Read / Write
<b>Allowed in motion:</b>	Yes
<b>Allowed with motor on:</b>	Yes
<b>Save to flash:</b>	No
<b>Axis related:</b>	No
<b>Min value:</b>	0
<b>Max value:</b>	10,000,000
<b>Default value:</b>	0
<b>User units:</b>	No user units
<b>Implementation status:</b>	Implemented

### Example:

AWaitTime, 100

### See Also:

Release Note:

- 1, (Done) 2018-07-24 update Douthport according to Eyal's email.
- 2, (Done) 2018-07-24 update PeakCL according to Eyal's email.  
This is implemented starting from FW version 1.2.0.1-6.
- 3, (Done) 2018-08-02 update UserPWM and DOutSelect.
- 4, (Done) 2018-08-16 update DouthSelect list for Centrali based on Eyal's email on 08-06.
- 5, (Done) 2018-08-21 update EmualRat information that it is useless with Central-i.
- 6, (TBD) 2018-08-23 two more commands "brakeused" and "regenused".  
Done on 08-24.
- 7, (Done) 2018-09-15 update HomingOn based on Eyal's email on 09-15.
- 8, (Done) 2018-09-28 update DInmode based on Eyal's email on 09-26.
- 9, (Done) 2018-09-28 update Setposition and homingon part based on Eyal's email about limitation of setposition.
- 10, (TB followed) 2018-10-19 update description about Amptype, maybe need more explanation.
- 11, (Done) Update DInMode according to Sophia's email on 20181223.
- 12, (Done) Update Math according to Eyal's email on 20190114.
- 13, (Done) Update ScheduleMode based on Eyal's email on 20190204.
- 14, (Done) Update StatReg based on Eyal's email on 20190204. TBD based on this email about motortemp and maxmotortemp.
- 15, (Done) Update ComtAng based on Eyal's email on 20190317.