BUCHER automation

Motion-API-Documentation



60888951_00

MCX API 2.0

Motion Control eXtended

This document has been compiled by Bucher Automation AG with due diligence based on the state of the art as known to them. Any revisions and technical advancements of our products are not automatically made available in a revised document.
Bucher Automation AG shall not be liable for any errors either in form or content, or for any missing updates, as well as for any damage or detriment resulting from such failure.

Bucher Automation AG
Thomas-Alva-Edison-Ring 10
71672 Marbach/Neckar, Germany
T +49 7141 2550-0
info@bucherautomation.com

Technical support
T +49 7141 2550-444
support@bucherautomation.com

Sales
T +49 7141 2550-663
sales@bucherautomation.com

www.bucherautomation.com

Translation of the german original User Manual

Revision	1.00
Date of issue	6/16/2026

Table of Contents

1 Introduction ... 26
1.1 Document Information ... 26
1.2 Typographical conventions ... 27
2 MCAxis.AxisConfigData ... 28
2.1 Syntax for MCAxis.AxisConfigData ... 28
2.2 Properties of MCAxis.AxisConfigData ... 28
2.2.1 DeviceName ... 28
2.2.2 AxisMode ... 29
2.2.3 AxisDriveType ... 29
2.2.4 ObjectID ... 30
2.2.5 Device ID ... 30
2.2.6 IsSimulation ... 30
3 MCAxis.Buffer ... 32
3.1 Syntax for MCAxis.Buffer ... 32
3.2 Functions of MCAxis.Buffer ... 32
3.2.1 Create() ... 32
3.2.2 Clear() ... 33
3.2.3 Delete() ... 34
3.2.4 Suspend() ... 35
3.2.5 Resume() ... 35
3.2.6 Wait() ... 36
3.3 Properties of MCAxis.Buffer ... 36
3.3.1 BufferState ... 37
3.3.2 IsCyclic ... 37
3.3.3 IsNormal ... 38
3.4 Example task for MCAxis.Buffer ... 38
4 MCAxis.Diagnostics ... 40
4.1 Syntax for MCAxis.Diagnostics ... 40
4.2 Features of MCAxis.Diagnostics ... 40
4.2.1 ClearErrors() ... 40
4.2.2 GetErrorCode() ... 41
4.2.3 GetMessageCode() ... 41
4.3 Properties of MCAxis.Diagnostics ... 42
4.3.1 ErrorCount ... 42
4.3.2 IsErrorPending ... 42
4.3.3 IsMessagePending ... 43

4.3.4 Message Count ... 43
4.4 Example task for MCAxis.Diagnostics ... 43
5 MCAxis.Drive.Amplifier ... 45
5.1 Syntax for MCAxis.Drive.Amplifier ... 45
5.2 Properties of MCAxis.Drive.Amplifier ... 45
5.2.1 Actual DC Link Voltage ... 45
5.2.2 Actual Heat Sink Temperature ... 46
5.2.3 Current Internal Temperature ... 46
5.2.4 DCLinkVoltageMax ... 46
5.2.5 DCLinkVoltageMin ... 47
5.2.6 PWM Frequency ... 47
5.2.7 PWM Frequency Reduction ... 47
5.2.8 Rated Current ... 48
5.2.9 Rated DC Link Voltage ... 48
5.2.10 WireTestMode ... 48
5.2.11 I2tValue ... 49
6 MCAxis.Drive.Brake ... 50
6.1 Syntax for MCAxis.Drive.Brake ... 50
6.2 Features of MCAxis.Drive.Brake ... 50
6.2.1 Apply() ... 50
6.2.2 Release() ... 50
6.3 Properties of MCAxis.Drive.Brake ... 51
6.3.1 ApplyDelay ... 51
6.3.2 ReleaseDelay ... 51
6.3.3 Mode ... 52
6.3.4 State ... 52
7 MCAxis.Drive.Control.Current ... 53
7.1 Syntax for MCAxis.Drive.Control.Current ... 53
7.2 Properties of MCAxis.Drive.Control.Current ... 53
7.2.1 Kp ... 53
7.2.2 Tn ... 54
7.2.3 Actual Motor Current ... 54
7.2.4 Actual Motor Voltage ... 54
7.2.5 Maximum Motor Current ... 55
8 MCAxis.Drive.Control.Position ... 56
8.1 Syntax for MCAxis.Drive.Control.Position ... 56
8.2 Functions of MCAxis.Drive.Control.Position ... 56
8.2.1 ResetTrailingIndicators() ... 56

8.3 Properties of MCAxis.Drive.Control.Position ... 56
8.3.1 Kv ... 57
8.3.2 FeedbackSource ... 57
8.3.3 SpeedFeedForwardScale ... 58
9 MCAxis.Drive.Control.Speed ... 59
9.1 Syntax for MCAxis.Drive.Control.Speed ... 59
9.2 Properties of MCAxis.Drive.Control.Speed ... 59
9.2.1 Kp ... 59
9.2.2 Tn ... 60
9.2.3 Tf ... 61
9.2.4 ActualValue ... 61
9.2.5 Setpoint ... 62
9.2.6 Isum ... 62
9.2.7 IsumPreset ... 62
9.2.8 Overload Factor ... 63
9.2.9 MaxMotorSpeed ... 63
9.2.10 Setpoint Limit ... 64
9.2.11 DifferenceWindow ... 64
9.2.12 TotalInertia ... 64
9.2.13 Torque Feedforward Scale ... 65
10 MCAxis.Drive.Control.Torque ... 66
10.1 Syntax for MCAxis.Drive.Control.Current ... 66
10.2 Properties of MCAxis.Drive.Control.Torque ... 66
10.2.1 ActualValue ... 66
10.2.2 Setpoint ... 67
10.2.3 Overload Factor ... 67
10.2.4 MaxMotorTorque ... 67
10.2.5 Setpoint Limit ... 68
11 MCAxis.Drive.Diagnostics ... 69
11.1 Syntax for MCAxis.Drive.Diagnostics ... 69
11.2 Features of MCAxis.Drive.Diagnostics ... 69
11.2.1 IsWarningPending() ... 69
11.3 Properties of MCAxis.Drive.Diagnostics ... 70
11.3.1 ActualError ... 70
11.3.2 IsErrorPending ... 71
12 MCAxis.Drive.DigitalInput ... 72
12.1 Syntax for MCAxis.Drive.DigitalInput ... 72
12.2 Properties of MCAxis.Drive.DigitalInput ... 72

12.2.1 Polarity Reference Switch ... 72
12.2.2 Limit Switch Mode ... 73
12.2.3 Polarity Limit Switch Min ... 73
12.2.4 PolarityLimitSwitchMax ... 74
12.2.5 IsPulseEnabled ... 74
13 MCAxis.Drive.EncoderChannel ... 75
13.1 Syntax for MCAxis.Drive.EncoderChannel ... 75
13.2 Properties of MCAxis.Drive.EncoderChannel ... 75
13.2.1 EncType ... 76
13.2.2 EncLines ... 76
13.2.3 Gearmotor ... 77
13.2.4 Gear Encoder ... 77
13.2.5 Multi-turn position ... 78
13.2.6 Single-Turn Position ... 78
13.2.7 Maximum Number of Absolute Revolutions ... 79
14 MCAxis.Drive.GenericAccess ... 80
14.1 Syntax for MCAxis.Drive.GenericAccess ... 80
14.2 Functions of MCAxis.Drive.GenericAccess ... 80
14.2.1 CANopen.ReadUInt8() ... 81
14.2.2 CANopen.WriteUInt8() ... 81
14.3 Properties of MCAxis.Drive.GenericAccess ... 82
14.3.1 CANopen OS Version ... 82
14.3.2 CANopen.IsAvailable ... 83
15 MCAxis.Drive.InvertDirection ... 84
15.1 Syntax for MCAxis.Drive.InvertDirection ... 84
15.2 Properties of MCAxis.Drive.InvertDirection ... 84
15.2.1 InvertDirection ... 84
16 MCAxis.Drive.IsReadyForOperation ... 85
16.1 Syntax for MCAxis.Drive.IsReadyForOperation ... 85
16.2 Properties of MCAxis.Drive.IsReadyForOperation ... 85
16.2.1 Ready for Operation ... 85
17 MCAxis.Drive.Motor ... 86
17.1 Syntax for MCAxis.Drive.Motor ... 86
17.2 Features of MCAxis.Drive.Motor ... 86
17.2.1 MeasureCommutationOffset() ... 86
17.3 Properties of MCAxis.Drive.Motor ... 87
17.3.1 Engine Design ... 88
17.3.2 Rated Current ... 89

17.3.3 MaxCurrent ... 90
17.3.4 Rated Speed ... 90
17.3.5 Rated Torque ... 91
17.3.6 Inductance ... 91
17.3.7 Resistance ... 92
17.3.8 VoltageConstant ... 93
17.3.9 Constant Torque ... 93
17.3.10 PolePairs ... 94
17.3.11 Commutation Offset ... 95
17.3.12 ThermalDelayTime ... 95
17.3.13 Shaft Position ... 96
17.3.14 Inertia ... 96
17.3.15 Temperature Sensor Type ... 97
17.3.16 Temperature Sensor Connector ... 98
17.3.17 Temperature ... 99
17.3.18 I2tOperatingMode ... 99
17.3.19 I2tValue ... 100

18 MCAxis.Drive.QuickStopDeceleration ... 101

18.1 Syntax for MCAxis.Drive.QuickStopDeceleration ... 101
18.2 Properties of MCAxis.Drive.QuickStopDeceleration ... 101
18.2.1 QuickStop Deceleration ... 101

19 MCAxis.Drive.TorqueShutOff ... 102

19.1 Syntax for MCAxis.Drive.TorqueShutOff ... 104
19.2 Features of MCAxis.Drive.TorqueShutOff ... 104
19.2.1 Configure() ... 104
19.2.2 ReadConfiguration() ... 105
19.2.3 Activate() ... 106
19.2.4 Deactivate() ... 107
19.3 Properties of MCAxis.Drive.TorqueShutOff ... 107
19.3.1 State ... 107
19.4 Example task MCAxis.Drive.TorqueShutOff ... 107

20 MCAxis.Info ... 109

20.1 Syntax for MCAxis.Info ... 109
20.2 Properties of the function ... 109
20.2.1 Name ... 109
20.2.2 ObjectID ... 109
20.2.3 ClassID ... 110
20.3 Example task MCAxis.Info ... 110

21 MCAxis.IsInTargetWindow 111
21.1 Syntax for MCAxis.IsInTargetWindow 111
21.2 Properties of MCAxis.IsInTartegWindow 111
21.2.1 IsInTargetWindow 111
21.3 Example task MCAxis.IsInTargetWindow 111
22 MCAxis.Mechanism.AxisMode 113
22.1 Syntax for MCAxis.Mechanism.AxisMode 114
22.2 Properties of MCAxis.Mechanism.AxisMode 114
22.2.1 AxisMode 114
23 MCAxis.Mechanism.AxisType 115
23.1 Syntax for MCAxis.Mechanism.AxisType 115
23.2 Properties of MCAxis.Mechanism.AxisType 115
23.2.1 AxisType 115
23.3 Description of Axis Types 116
23.3.1 Rotational axis 116
23.3.2 Linear axis 117
23.3.3 Non-linear eccentric axis 118
23.3.4 Non-linear piston axis 118
23.4 Example task for MCAxis.Mechanism.AxisType 119
24 MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor 120
24.1 Syntax for MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor 120
24.2 Properties of MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor 120
24.2.1 Gear Ratio Motor 120
24.2.2 Gear Ratio Load 121
25 MCAxis.Mechanism.Kinematics 122
25.1 Syntax for MCAxis.Mechanism.Kinematics 122
25.2 Functions of MCAxis.Mechanism.Kinematics 122
25.2.1 Crank.Configure(), Piston.Configure() 122
25.3 Properties of MCAxis.Mechanism.Kinematics 123
25.3.1 .Offset / .InvertDirection / .Radius / .RodLength 123
25.4 Example task MCAxis.Mechanism.Kinematics 124
26 MCAxis.Mechanism.Limitations 125
26.1 Syntax for MCAxis.Mechanism.Limitations 125
26.2 Functions of MCAxis.Mechanism.Limitations 125
26.2.1 SetTravelLimits() 125
26.3 Properties of MCAxis.Mechanism.Limitations 126
26.3.1 TravelLimitMin 126
26.3.2 TravelLimitMax 127

26.3.3 MaxSpeed 128
26.3.4 MaxAcceleration 128
26.3.5 MaxJerk 129

27 MCAxis.Mechanism.LinearRotationRatio 130

27.1 Syntax for MCAxis.Mechanism.LinearRotationRatio 130
27.2 Properties of MCAxis.Mechanism.LinearRotationRatio 130
27.2.1 Linear-to-Rotational Ratio 130

28 MCAxis.Mechanism.Monitoring 132

28.1 Syntax for MCAxis.Mechanism.Monitoring 132
28.2 Properties of MCAxis.Mechanism.Monitoring 132
28.2.1 IsReferenceSwitchEnabled 132
28.2.2 IsHWLimitSwitchMinActive 133
28.2.3 IsHWLimitSwitchMaxActive 133

29 MCAxis.Mechanism.Slope 134

29.1 Syntax for MCAxis.Mechanism.Slope 134
29.2 Properties of MCAxis.Mechanism.Slope 134
29.2.1 SlopeState 135
29.2.2 IsAccelerating 135
29.2.3 IsAtConstantSpeed 135
29.2.4 IsDecelerating 136
29.2.5 IsStopped 136
29.2.6 Slope Type 136
29.2.7 ActualMaxSpeed 137
29.2.8 Maximum Acceleration 137
29.2.9 Maximum Deceleration 138
29.2.10 Acceleration Setpoint 138
29.2.11 SpeedSetpoint 138
29.2.12 Speed 139
29.2.13 Acceleration 139
29.2.14 Deceleration 140
29.2.15 Config.RampBreakMode 140

30 MCAxis.Mechanism.Units 141

30.1 Syntax for MCAxis.Mechanism.Units 141
30.2 Functions of MCAxis.Mechanism.Units 141
30.2.1 SetLinearUnits() 141
30.2.2 SetRotaryUnits() 142
30.3 Properties of MCAxis.Mechanism.Units 143
30.3.1 LinearPositionUnit 143

30.3.2 LinearSpeedUnit 144
30.3.3 LinearAccelerationUnit 144
30.3.4 LinearJerkUnit 145
30.3.5 Rotary Position Unit 145
30.3.6 Rotary Speed Unit 146
30.3.7 Rotary Acceleration Unit 146
30.3.8 RotatoryJerkUnit 146
31 MCAxis.MoveHalt 148
31.1 Syntax for MCAxis.MoveHalt 148
31.2 Functions of MCAxis.MoveHalt 148
31.2.1 Start() 148
31.3 Example Task: MCAxis.MoveHalt 150
32 MCAxis.MoveHome 151
32.1 Syntax for MCAxis.MoveHome 151
32.2 Functions of MCAxis.MoveHome 151
32.2.1 Start() 151
32.2.2 SetReference() 153
32.2.3 SetAbsEncoderReference() 154
32.2.4 ClearAbsEncoderReferenceStatus() 155
32.3 Properties of MCAxis.MoveHome 155
32.3.1 Directions 156
32.3.2 SwitchConfig 157
32.3.3 Mode 158
32.3.4 Acceleration 159
32.3.5 SpeedSwitchSearch 159
32.3.6 MaxDistanceSwitchSearch 160
32.3.7 SpeedReferenceSearch 161
32.3.8 MaxDistanceReferenceSearch 161
32.3.9 Reference Offset 162
32.3.10 Normal Position 163
32.3.11 ReferenceState 163
32.3.12 IsReferenceRunStarted, IsSwitchFound, IsReferenceFound, IsReferenceSet 164
32.3.13 IsAbsEncoderReferenceValid 165
32.3.14 ReferenceShift 165
33 MCAxis.MovePtp 167
33.1 Syntax for MCAxis.MovePtp 167
33.2 Functions of MCAxis.MovePtp 167
33.2.1 Start() 167

33.2.2 StartNewTargetPosition() 170
33.2.3 NewSpeed() 172
33.2.4 NewAcceleration() 173
33.2.5 NewDeceleration() 174
33.3 Properties of MCAxis.MovePtp 175
33.3.1 Positioning Mode 175
33.3.2 TargetPosition 177
33.3.3 TargetWindow 177
33.4 Example task MCAxis.MovePtp 179
34 MCAxis.MoveVelocity 181
34.1 Syntax for MCAxis.MoveVelocity 181
34.2 Functions of MCAxis.MoveVelocity 181
34.2.1 Start() 181
34.2.2 Reverse() 184
34.2.3 NewSpeed() 184
34.2.4 NewAcceleration() 185
34.2.5 NewDeceleration() 186
34.3 Properties of MCAxis.MoveVelocity 187
34.3.1 Directions 187
34.4 Example task MCAxis.MoveVelocity 188
35 MCAxis.Position 189
35.1 Syntax for MCAxis.Position 189
35.2 Functions of MCAxis.Position 189
35.2.1 AbsEncOverflowHandler.Activate() 189
35.2.2 AbsEncOverflowHandler.Deactivate() 190
35.3 Properties of MCAxis.Position 190
35.3.1 Target 191
35.3.2 Actual 191
35.3.3 Setpoint 191
35.3.4 Encoder Position 192
35.3.5 Current Tracking Error 192
35.3.6 TrailingIndicatorMax 192
35.3.7 TrailingIndicatorMin 193
35.3.8 AbsEncOverflowHandler.IsActive 193
36 MCAxis.Power 194
36.1 Syntax for MCAxis.Power 194
36.2 Features of MCAxis.Power 194
36.2.1 Enable() 194

36.2.2 Disable() 195
36.2.3 QuickStop() 196
36.3 Example task: MCAxis.Power 197
37 MCAxis.Settings 198
37.1 Syntax for MCAxis.Settings 198
37.2 Features of MCAxis.Settings 198
37.2.1 SaveToFile() 198
37.2.2 LoadFromFile() 199
38 MCAxis.State 200
38.1 Syntax for MCAxis.State 202
38.2 Properties of MCAxis.State 203
38.2.1 Value 203
38.2.2 IsDisabled 203
38.2.3 IsEnabled 204
38.2.4 IsRunning 204
38.2.5 IsSuspended 204
38.2.6 IsDisabledError 205
38.2.7 IsEnabledError 205
38.2.8 IsInactive 205
38.2.9 OperatingMode.Value 205
38.2.10 TypeOfMotion.Value 208
38.3 Example task MCAxis.State 211
39 MCAxis.State.Techno 212
39.1 Syntax for MCAxis.State.Techno 212
39.2 Properties of MCAxis.State.Techno 212
39.2.1 TechnoID 212
39.2.2 SyncState 213
39.2.3 IsDecoupled 213
39.2.4 IsDecoupling 213
39.2.5 IsCoupling 214
39.2.6 IsCoupled 214
39.2.7 SyncPosition 214
39.2.8 Cam.CamID 215
39.2.9 Cam.CamSegmentID 215
39.2.10 Cam.MasterOffset 216
39.2.11 Cam.FollowerOffset 216
39.2.12 Cam.MasterFactor 217
39.2.13 Cam.FollowerFactor 218

40 MCAxis.State.Transitions 219
40.1 Syntax for MCAxis.State.Transitions 221
40.2 Functions of MCAxis.State.Transitions 222
40.2.1 Enable() 222
40.2.2 Disable() 222
40.2.3 ErrorDisable() 222
40.2.4 ErrorStop() 223
40.2.5 ErrorLockUserInput() 224
40.2.6 Suspend() 224
40.2.7 Resume() 225
40.2.8 UserPositionReset() 226
40.3 Properties of MCAxis.State.Transitions 226
40.3.1 ErrorDisableMode 227
40.3.2 ErrorStopMode 227
41 MCAxis.Stop 228
41.1 Features of MCAxis.Stop 228
41.1.1 Stop() 228
41.2 Example task: MCAxis.Stop 229
42 MCAxis.TouchProbe 231
42.1 Syntax for MCAxis.TouchProbe 234
42.2 Features of MCAxis.TouchProbe 234
42.2.1 Configure() 234
42.2.2 Enable() 235
42.2.3 Disable() 236
42.2.4 ReadConfiguration() 236
42.2.5 ReadState() 237
42.3 Example task: MCAxis.TouchProbe 239
43 MCCouplingGroup 241
43.1 Syntax for MCCouplingGroup 241
43.2 Features of MCCouplingGroup 241
43.2.1 AddFollowerAxis() 242
43.2.2 Redefine() 242
43.2.3 Undefine() 243
43.3 Properties of the MCCouplingGroup 243
43.3.1 Follower Count 243
43.3.2 MasterAxis 244
43.3.3 FollowerAxis[] 244
43.3.4 Name 244

44 MCGantryGroup 246
44.1 Syntax for MCGantryGroup 246
44.2 Features of MCGantryGroup 247
44.2.1 AddFollowerAxis() 247
44.2.2 Redefine() 247
44.2.3 Undefine() 248
44.3 Properties of MCGantryGroup 248
44.3.1 GantryWindow[i] 249
44.3.2 ConfigGantryWindow[i] 249
44.3.3 Follower Count 249
44.3.4 MasterAxis 250
44.3.5 FollowerAxis[i] 250
44.3.6 Name 250
44.4 Example Task: MCGantryGroup 251
45 MCGeo.Activate, MCGeo.Deactivate 253
45.1 Syntax for MCGeo.Activate, MCGeo.Deactivate 253
45.2 Functions of MCGeo.Activate and MCGeo.Deactivate 253
45.2.1 Activate() 253
45.2.2 Deactivate() 254
46 MCGeo.Buffer 256
46.1 Syntax for MCGeo.Buffer 256
46.2 Functions of MCGeo.Buffer 256
46.2.1 Clear() 256
46.2.2 Suspend() 257
46.2.3 Resume() 258
46.2.4 UserIndex.Enable() 258
46.2.5 UserIndex.Disable() 259
46.2.6 UserIndex.Reset() 259
46.3 Properties of MCGeo.Buffer 260
46.3.1 UserIndex.Actual 260
47 MCGeo.Coordinates 261
47.1 Syntax for MCGeo.Coordinates 261
47.2 Properties of MCGeo.Coordinates 261
47.2.1 Actual 261
47.2.2 Setpoint 262
47.2.3 Target 262
48 MCGeo.Coupling 264
48.1 Syntax for MCGeo.Coupling 265

48.2 Features of MCGeo.Coupling 265
48.2.1 Couple() 265
48.2.2 Decouple() 267
48.2.3 GetPositioningMode() 268
48.2.4 GetCoupleType() 268
48.2.5 GetCoupleMode() 268
48.2.6 GetDecoupleMode() 269
48.3 Properties of MCGeo.Coupling 269
48.3.1 CoupleState 269
49 MCGeo.Diagnostics 270
49.1 Syntax for MCGeo.Diagnostics 270
49.2 Features of MCGeo.Diagnostics 270
49.2.1 ClearErrors(), Software.ClearErrors() – path group 270
49.2.2 Software.GetErrorCode() 271
49.2.3 Software.GetMessageCode() 271
49.3 Properties of MCGeo.Diagnostics 273
49.3.1 Software.ErrorCount 273
49.3.2 IsErrorPending, Software.IsErrorPending 273
49.3.3 Software.MessageCount 273
49.3.4 IsMessagePending, Software.IsMessagePending 274
49.4 Example task for MCGeo.Diagnostics 274
50 MCGeo.Events 276
50.1 Syntax for MCGeo.Events 276
50.2 Features of MCGeo.Events 276
50.2.1 SetFlag() 276
50.2.2 ReadFlag() 277
50.2.3 ReadFlags() 278
50.2.4 ClearFlags() 278
50.3 Example task: MCGeo.Events 278
51 MCGeo.Info 280
51.1 Syntax for MCGeo.Info 280
51.2 Properties of the function 280
51.2.1 Name 280
51.2.2 ObjectID 280
51.2.3 ClassID 281
51.3 Example task: MCGeo.Info 281
52 MCGeo.Mechanism.Kinematics 282
52.1 Syntax for MCGeo.Mechanism.Kinematics 282

52.2 Properties of MCGeo.Mechanism.Kinematics 282
52.2.1 Cartesian 2D 282
52.2.2 Cartessian3D 283
52.2.3 Scara 284
52.3 Configuration features of MCGeo.Mechanism.Kinematics 285
52.3.1 Cartesian2D.X.Configure(), Cartesian2D.Y.Configure() 285
52.3.2 Cartesian3D.X.Configure(), Cartesian3D.Y.Configure() 286
52.3.3 Scara.Shoulder.Configure(), Scara.Elbow.Configure() 287
52.3.4 Scara.Vertical.Configure() 288
52.3.5 Scara.Wrist.Configure() 288
53 MCGeo.Mechanism.Limitations 290
53.1 Syntax for MCGeo.Mechanism.Limitations 291
53.2 Properties of MCGeo.Mechanism.Limitations 291
53.2.1 WorkspaceLimitMin 292
53.2.2 WorkspaceLimitMax 293
53.2.3 OrientationLimitMin 293
53.2.4 OrientationLimitMax 294
53.2.5 OrientationFilter 295
53.2.6 MaxPathVelocity 296
53.2.7 MaxPathAcceleration 296
53.2.8 MaxPathJerk 297
53.2.9 SingularityDistance1 298
54 MCGeo.Mechanism.Slope 299
54.1 Syntax for MCGeo.Mechanism.Slope 299
54.2 Properties of MCGeo.Mechanism.Slope 299
54.2.1 SlopeState 300
54.2.2 IsAccelerating 300
54.2.3 IsAtConstantSpeed 300
54.2.4 IsDecelerating 301
54.2.5 IsStopped 301
54.2.6 ActualMaxSpeed 301
54.2.7 Maximum Acceleration 302
54.2.8 SpeedSetpoint 302
54.2.9 Acceleration Setpoint 302
54.2.10 Speed 303
54.2.11 MaxVeloPoly 303
54.2.12 SpeedOverride 303
54.2.13 Acceleration 304

55 MCGeo.Mechanism.Units 305
55.1 Syntax for MCGeo.Mechanism.Units 305
55.2 Functions of MCGeo.Mechanism.Units 305
55.2.1 SetUnits() 305
55.3 Properties of MCGeo.Mechanism.Units 307
55.3.1 PositionUnit 307
55.3.2 Orientation Unit 307
55.3.3 SpeedUnit 308
55.3.4 AccelerationUnit 308
55.3.5 JerkUnit 309
56 MCGeo Members 310
56.1 Syntax for MCGeo.Members 310
56.2 Features of MCGeo.Members 310
56.2.1 AddSyncAxis() 310
56.2.2 RemoveSyncAxis() 311
57 MCGeo.MoveCircle2D 312
57.1 Syntax for MCGeo.MoveCircle2D 312
57.2 Functions of MCGeo.MoveCircle2D 312
57.2.1 Start() 313
57.3 Properties of MCGeo.MoveCircle2D 315
57.3.1 Positioning Mode 315
57.3.2 BlockEnd 316
57.3.3 Coordinate System Plane 317
57.3.4 TravelAngle 318
57.3.5 TargetPositionHelixAxis 318
57.3.6 Smoothing Distance 318
57.3.7 Delay 319
57.3.8 TargetWindow 319
57.3.9 CenterPoint 319
57.3.10 Target Orientation 320
58 MCGeo.MoveCircle3D 321
58.1 Syntax for MCGeo.MoveCircle3D 322
58.2 Functions of MCGeo.MoveCircle3D 322
58.2.1 Start() 322
58.3 Properties of MCGeo.MoveCircle3D 324
58.3.1 Positioning Mode 324
58.3.2 BlockEnd 325
58.3.3 Smoothing Distance 326

58.3.4 Delay 326
58.3.5 TargetWindow 327
58.3.6 Intermediate Point 327
58.3.7 TargetPosition 328
58.3.8 Target Orientation 328
59 MCGeo.MoveLinear 329
59.1 Syntax for MCGeo.MoveLinear 329
59.2 Functions of MCGeo.MoveLinear 329
59.2.1 Start() 329
59.3 Properties of MCGeo.MoveLinear 331
59.3.1 Positioning Mode 331
59.3.2 BlockEnd 332
59.3.3 Smoothing Distance 333
59.3.4 Delay 333
59.3.5 TargetWindow 334
59.3.6 Target Coordinates 334
59.3.7 TargetPosition 335
59.3.8 Target Orientation 335
60 MCGeo.MoveOrientation 337
60.1 Syntax for MCGeo.MoveOrientation 337
60.2 Functions of MCGeo.MoveOrientation 337
60.2.1 Start() 337
60.3 Properties of MCGeo.MoveOrientation 338
60.3.1 Positioning Mode 339
60.3.2 BlockEnd 340
60.3.3 Speed 340
60.3.4 Acceleration 341
60.3.5 Jerk 341
60.3.6 Delay 341
60.3.7 TargetWindow 342
60.3.8 Target Orientation 342
61 MCGeo.MovePtp 343
61.1 Syntax for MCGeo.MovePtp 343
61.2 Features of MCGeo.MovePtp 343
61.2.1 Start() 343
61.3 Properties of MCGeo.MovePtp 345
61.3.1 Positioning Mode 345
61.3.2 BlockEnd 346

61.3.3 Delay 347
61.3.4 TargetWindow 347
61.3.5 TargetPosition 347
61.3.6 Target Orientation 348
61.3.7 Target Coordinates 348
62 MCGeo.MoveSpline 350
62.1 Syntax for MCGeo.MoveSpline 350
62.2 Functions of MCGeo.MoveCircle2D 350
62.2.1 Start() 350
62.3 Properties of MCGeo.MoveSpline 353
62.3.1 NodeCount 353
62.3.2 Positioning Mode 353
62.3.3 BlockEnd 354
62.3.4 Start Condition 355
62.3.5 TargetCondition 356
62.3.6 Smoothing Distance 357
62.3.7 Delay 357
62.3.8 TargetWindow 357
62.3.9 VirtualStartPosition[x] 358
62.3.10 VirtualTargetPosition[x] 358
62.3.11 NodePosition[n] 359
62.3.12 NodeOrientation[n] 359
62.3.13 NodeCoordinates[n] 360
63 MCGeo.MoveSyncAxis 361
63.1 Syntax for MCGeo.MoveSyncAxis 361
63.2 Functions of MCGeo.MoveSyncAxis 361
63.2.1 Start() 361
63.2.2 GetTargetPosition() 362
63.2.3 SetTargetPosition() 363
63.2.4 GetPositioningMode() 363
63.2.5 SetPositioningMode() 364
64 MCGeo.Orientation 365
64.1 Syntax for MCGeo.Orientation 365
64.2 Properties of MCGeo.Orientation 365
64.2.1 Actual 365
64.2.2 Setpoint 366
64.2.3 Target 367
65 MCGeo.Position 368

65.1 Syntax for MCGeo.Position 368
65.2 Properties of MCGeo.Position 368
65.2.1 Actual 368
65.2.2 Setpoint 369
65.2.3 Target 369
66 MCGeo.Power 370
66.1 Syntax for MCGeo.Power 370
66.2 Features of MCGeo.Power 370
66.2.1 Enable() 370
66.2.2 Disable() 371
66.2.3 QuickStop() 372
66.3 Example task: MCGeo.Power 373
67 MCGeo.Settings 374
67.1 Syntax for MCGeo.Settings 374
67.2 Features of MCGeo.Settings 374
67.2.1 SaveToFile() 374
67.2.2 LoadFromFile() 375
68 MCGeo.State 376
68.1 Properties of MCGeo.State 378
68.1.1 Value 379
68.1.2 IsDisabled 379
68.1.3 IsDisabledError 379
68.1.4 IsEnabled 380
68.1.5 IsEnabledError 380
68.1.6 IsInactive 380
68.1.7 IsRunning 381
68.1.8 IsSuspended 381
68.1.9 OperatingMode.Value 381
68.1.10 TypeOfMotion.Value 383
68.2 Example task: MCGeo.State 386
69 MCGeo.State.Transitions 387
69.1 Syntax for MCGeo.State.Transitions 389
69.2 Functions of MCGeo.State.Transitions 389
69.2.1 Enable() 390
69.2.2 Disable() 390
69.2.3 Stop() 390
69.2.4 ErrorDisable() 392
69.2.5 ErrorStop() 392

69.2.6 ErrorLockUserInput() 393
69.2.7 Suspend() 393
69.2.8 ErrorSuspend() 394
69.2.9 Resume() 395
69.3 Properties of MCGeo.State.Transitions 395
69.3.1 ErrorDisableMode 396
69.3.2 ErrorStopMode 397
69.3.3 ErrorLevel 397
70 MCGeo.Stop 399
70.1 Syntax for MCGeo.Stop 399
70.2 Features of MCGeo.Stop 399
70.2.1 Stop() 399
71 MCManager (MCMgr) 401
71.1 Syntax for MCMgr 401
71.2 Features of MCMgr 401
71.2.1 CheckVersions() 401
71.2.2 GetDataSource() 402
71.2.3 WriteLogToFile() 402
71.2.4 GetAxisByID() 403
71.2.5 GetAxisByName() 403
71.2.6 GetTechnoByID() 403
71.2.7 GetTechnoByName() 404
71.2.8 GetGeoByID() 404
71.2.9 GetTechnoByName() 404
71.3 Properties of MCMgr 405
71.3.1 AxisCount 405
71.3.2 AxisList[] 405
71.3.3 TechnoCount 406
71.3.4 TechnoList[] 406
71.3.5 GeoCount 406
71.3.6 GeoList[] 407
71.3.7 Number of Faulty Objects 407
71.3.8 FaultyObjectList[] 407
71.3.9 Name 409
71.3.10 Version 409
71.3.11 IsInitialized 409
72 MCTechno.Activate, MCTechno.Deactivate 410
72.1 Syntax for MCTechno.Activate, MCTechno.Deactivate 410

72.2 Functions of MCTechno.Activate and MCTechno.Deactivate 410
72.2.1 Activate() 410
72.2.2 Deactivate() 411
73 MCTechno.Coupling 413
73.1 Syntax for MCTechno.Coupling 413
73.2 Features of MCTechno.Coupling 413
73.2.1 Couple() 414
73.2.2 Decouple() 415
73.2.3 GetCoupleMasterPosition() 416
73.2.4 GetCoupleFollowerPosition() 417
73.2.5 GetCoupleMasterDirection() 417
73.2.6 GetDecoupleDeceleration() 418
73.2.7 GetDecoupleMasterPosition() 418
73.2.8 GetDecoupleMode() 418
73.2.9 IsCoupling() 419
73.2.10 IsCoupled() 419
73.2.11 IsDecoupling() 419
73.2.12 IsDecoupled() 420
73.2.13 Filter.GetCoefficient(), Filter.SetCoefficient() 420
73.2.14 Gearing.Configure() 421
73.2.15 Gearing.GetFactor() 422
73.2.16 Gearing.GetDivisor() 422
73.2.17 Gearing.GetActivationMode() 423
73.3 Properties of MCTechno.Coupling 423
73.3.1 ExternalMaster.Offset 423
73.3.2 ExternalMaster.Smoothing 424
74 MCTechno.Coupling.Cam 426
74.1 Syntax for MCTechno.Coupling.Cam 426
74.2 Features of MCTechno.Coupling.Cam 426
74.2.1 Create() 426
74.2.2 DefineSegment() 427
74.2.3 DefineSegmentAuto() 429
74.2.4 Activate() 430
74.2.5 ReadSegment() 431
74.2.6 Evaluate() 433
74.2.7 AnalyzeSegment() 433
74.2.8 AnalyzeDerivatives() 435
75 MCTechno.Diagnostics 438

75.1 Syntax for MCTechno.Diagnostics 438
75.2 Features of MCTechno.Diagnostics 438
75.2.1 ClearErrors(), Software.ClearErrors() – technology group 438
75.2.2 Software.GetErrorCode() 439
75.2.3 Software.GetMessageCode() 439
75.3 Properties of MCTechno.Diagnostics 440
75.3.1 Software.ErrorCount 440
75.3.2 IsErrorPending, Software.IsErrorPending 440
75.3.3 Software.MessageCount 441
75.3.4 IsMessagePending, Software.IsMessagePending 441
75.4 Example task for MCTechno.Diagnostics 441
76 MCTechno.Info 443
76.1 Syntax for MCTechno.Info 443
76.2 Properties of the function 443
76.2.1 Name 443
76.2.2 ObjectID 443
76.2.3 ClassID 444
76.3 Example Task MCTechno.Info 444
77 MCTechno.Mechanism.Slope 445
77.1 Syntax for MCTechno.Mechanism.Slope 445
77.2 Functions of MCTechno.Mechanism.Slope 445
77.2.1 GetSlopeType(), SetSlopeType() 445
77.2.2 GetSpeed(), SetSpeed() 446
77.2.3 GetAcceleration(), SetAcceleration() 447
77.2.4 GetDeceleration(), SetDeceleration() 447
77.3 Properties of MCTechno.Mechanism.Slope 448
77.3.1 SlopeState 448
77.3.2 IsAccelerating 449
77.3.3 IsAtConstantSpeed 449
77.3.4 IsDecelerating 449
77.3.5 IsStopped 449
78 MCTechno Members 451
78.1 Syntax for MCTechno.Members 451
78.2 Properties of MCTechno.Members 451
78.2.1 Follower Count 451
78.2.2 Follower[s] 451
78.2.3 Follower Count 452
78.2.4 FollowerCam[n] 452

78.2.5 FollowerGearCount 452
78.2.6 FollowerGear[n] 453
78.2.7 Master's 453
79 MCTechno.MoveHalt 454
79.1 Syntax for MCTechno.MoveHalt 454
79.2 Features of MCTechno.MoveHalt 454
79.2.1 Start() 454
80 MCTechno.MovePtp 457
80.1 Syntax for MCTechno.MovePtp 457
80.2 Features of MCTechno.MovePtp 457
80.2.1 Start() 457
80.2.2 StartNewTargetPosition() 460
80.2.3 NewSpeed() 461
80.2.4 NewAcceleration() 462
80.2.5 NewDeceleration() 463
80.2.6 GetPositioningMode(), SetPositioningMode() 464
80.2.7 GetTargetPosition(), SetTargetPosition() 466
80.2.8 GetTargetWindow(), SetTargetWindow() 467
80.2.9 StartSuperPose() 468
80.2.10 StartNewTargetPositionSuperPose() 469
81 MCTechno.MoveVelocity 471
81.1 Syntax for MCTechno.MoveVelocity 471
81.2 Functions of MCTechno.MoveVelocity 471
81.2.1 Start() 471
81.2.2 Reverse() 474
81.2.3 NewSpeed() 475
81.2.4 NewAcceleration() 476
81.2.5 NewDeceleration() 477
81.2.6 GetDirection(), SetDirection() 478
81.2.7 StartSuperPose() 479
81.2.8 ReverseSuperPose() 481
82 MCTechno.Power 483
82.1 Syntax for MCTechno.Power 483
82.2 Features of MCTechno.Power 483
82.2.1 Enable() 483
82.2.2 Disable() 484
82.2.3 QuickStop() 485
82.3 Example Task: MCTechno.Power 486

83 MCTechno.Settings 487
83.1 Syntax for MCTechno.Settings 487
83.2 Features of MCTechno.Settings 487
83.2.1 SaveToFile() 487
83.2.2 LoadFromFile() 488
84 MCTechno.State 489
84.1 Syntax for MCTechno.State 491
84.2 Properties of MCTechno.State 491
84.2.1 Value 492
84.2.2 IsDisabled 492
84.2.3 IsDisabledError 492
84.2.4 IsEnabled 493
84.2.5 IsEnabledError 493
84.2.6 IsInactive 493
84.2.7 IsRunning 494
84.2.8 IsSuspended 494
84.2.9 OperatingMode.Value 494
84.2.10 TypeOfMotion.Value 495
85 MCTechno.State.Transitions 498
85.1 Syntax for MCTechno.State.Transitions 500
85.2 Functions of MCTechno.State.Transitions 500
85.2.1 Enable() 501
85.2.2 Disable() 501
85.2.3 Stop() 501
85.2.4 ErrorDisable() 503
85.2.5 ErrorStop() 503
85.2.6 ErrorLockUserInput() 504
85.2.7 Suspend() 504
85.2.8 Resume() 505
85.3 Properties of MCTechno.State.Transitions 506
85.3.1 ErrorDisableMode 506
85.3.2 ErrorStopMode 507
85.3.3 ErrorLevel 507
86 MCTechno.Stop 509
86.1 Syntax for MCTechno.Stop 509
86.2 Features of MCTechno.Stop 509
86.2.1 Stop() 509
Glossary 511

1 Introduction

1.1 Document Information

This document is part of the product and must be read and understood before using the device. It contains important and safety-related information to ensure that the product is used properly and in accordance with its intended purpose.

Purpose of the Document

This document describes rules for the design of the system menu in ISOBUS operating devices of Bucher Automation AG.

It does not describe all the functions and operation of the virtual ISOBUS terminal.

Target audiences

This document is intended for qualified personnel.
The device must only be put into operation by qualified and trained personnel.
Safe use of the device must be ensured throughout every stage of its life cycle. A lack of or insufficient technical and documentation knowledge will result in the loss of any liability claims.

Availability of information

Make sure this document is kept at the ready in the vicinity of the product throughout its service life.
Check this document regularly for updates and revisions. This document is not subject to any updating service.

www.bucherautomation.cn

For further information refer to the following information products:

– Application Notes
  Technical reports and application examples
– User manuals
  Information on commissioning Bucher Automation products
– Programming Manual

  For the programming and parameterization of the product, it is imperative that the *JSCM programming manual* be observed.
– Programming Manuals
  Application-oriented programming examples
– JetSymSoftware Online Help: Software
  Product Features with Application Examples
– Application-oriented manuals
  Product-independent documentation
– Version
  Updates: Information about changes to the software products and your device's operating system
– OEM manual
  Diagnostics and software requirements according to ISO 13849-1
– How-to manual

  Description of the programming and update workflows.

NOTICE



Applicable documents

This document does not replace the user manual for JM-2000 servo amplifiers. It supplements these manuals with a description of safety function S1 (STO and SBC).

ⓘ Info

Further information

For further information on the noise immunity of a system, please refer to the Application Note 016 *EMC-Compatible Installation of the Electric Cabinet* available for download on www.bucherautomation.com.

ⓘ Info

EtherCAT®

EtherCAT® is a registered trademark and patented technology licensed by Beckhoff Automation GmbH, Germany.

1.2 Typographical conventions

This manual uses different typographical effects to support you in finding and classifying information. Below, there is an example of a step-by-step instruction:

✓ This symbol indicates requirements which have to be met before executing the following action.

► This sign or a numbering at the beginning of a paragraph marks an action instruction that must be executed by the user. Execute the instructions one after the other.

⇨ The target after a list of instructions indicates reactions to, or results of these actions.

ⓘ Info

Further information and practical tips

In the info box you will find helpful information and practical tips about your product.

2 MCAxis.AxisConfigData

The **AxisConfigData** function allows you to retrieve various pieces of information about an MCX axis based on the configuration in the JetSym programming tool.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

2.1 Syntax for MCAxis.AxisConfigData

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.AxisConfigData.<Access to Value>;
```

2.2 Properties of MCAxis.AxisConfigData

This function offers the following properties:

- DeviceName
- AxisMode
- AxisDriveType
- ObjectID
- DeviceID
- IsSimulation

2.2.1 DeviceName

Description

data type	Meaning
String	Reads the name assigned to this MCX axis in JetSym . This name also refers to the axis object created by JetSym for programming purposes.

Syntax

```
AxisName := AX1.AxisConfigData.DeviceName;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

2.2.2 AxisMode

Description

data type	Meaning
Type MCAxisModes	Defines the axis type.
MCAxisModes.InternalAxis	A fully functional, real-world axis with a servo amplifier and motor.
MCAxisModes.ExternalAxis	An axis that has only a current position. It can be used as a central axis in a technological group. It is defined by the second encoder on a physical axis.
MCAxisModes.SimulatedAxis	An axis that behaves like an internal axis, but in which the servo amplifier and motor are simulated. This configuration allows you to develop and test a program even when the actual hardware components are not yet available.
MCAxisModes.VirtualAxis	Axis calculated in the MCX kernel. It serves as an ideal guiding principle for technology groups.
MCAxisModes.ShadowAxis	A drive shaft primarily used to generate cascaded techno groups. It always assumes the target and actual positions of another (virtual or real) axis. A driven axis must always be the master axis in a technology group.

Syntax

```
AxisType := AX1.AxisConfigData.AxisMode;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

2.2.3 AxisDriveType

Description

Use this property to read the basic attributes of the MCX axis. This information is used to determine, for loop commands across all axes, whether a command makes sense for that axis or not. For example, for axes with `MCAxisDriveType.None`, it makes no sense to read or write parameters that relate to the servo amplifier.

data type	Meaning
MCAxisDriveTypes	Defines the axis type.
MCAxisDriveTypes.Managed	Real axis (`InternalAxis`), external axis (`ExternalAxis`), simulated (real) axis (`SimulatedAxis`).
MCAxisDriveTypes.None	Virtual axis, shadow axis (`ShadowAxis`)

Syntax

```
Axis Type := AX1.AxisConfigData.AxisDriveType;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them. The state of the axis remains unchanged.

The property can be read in any state.

2.2.4 ObjectID

Description

data type	Meaning
Integer	Reads the internal object number of the MCX axis relative to all axis objects of the MCX axes created in the JetSymHardware Manager. Each axis object is assigned a unique number.

Syntax

```
AxisObjectID := AX1.AxisConfigData.ObjectID;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them. The state of the axis remains unchanged.

The property can be read in any state.

2.2.5 DeviceID

Description

data type	Meaning
Int	Reads the internal device number of the MCX axis relative to the bus system. This number is assigned by the JetSymHardware Manager for internal communication. Each bus participant is assigned a unique DeviceID. Axes that are not bus participants (e.g., virtual axes) return the value `-1` .

Syntax

```
BusObjectID := AX1.AxisConfigData.DeviceID;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them. The state of the axis remains unchanged.

The property can be read in any state.

2.2.6 IsSimulation

Description

data type	Meaning
Bool	Checks whether the MCX axis has been configured for simulation. In projects where not every machine uses an MCX axis, this property makes it possible to determine within the program whether the axis in question is used in the configuration for that particular machine. `True`: The MCX axis is operated in simulation mode.

Syntax

```
If AX1.State isnot "Simulation", then
...
End_If;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them. The state of the axis remains unchanged.

The property can be read in any state.

3 MCAxis.Buffer

The **MCAxis.Buffer** function allows you to switch the axis between buffered and normal operation. In addition, you can use the properties and functions of this function to manage the buffer and check the buffer status.

ⓘ Info

axis behavior in buffered operation

Configuration and parameterization of an axis are disabled in buffered mode.

After power-up, the axis is in the *normal*operating state, in which motion commands are "displacement-based." This means that a new motion command immediately replaces the previously active motion.

In the *buffered* operating state, motion commands for the axis are written to an internal MCX buffer and executed sequentially. The condition for transitioning from one motion to the next (block change condition) depends on the type of motion command. Specifically, the following conditions for changing sets apply:

Movement command	Condition
MCAxis.MovePtp()	Reaching the target position of the setpoint (interpolator stop).
MCAxis.MoveVelocity()	Reaching the target speed.
MCAxis.MoveHome()	Reaching the reference position of the setpoint (interpolator stop).
MCAxis.MoveHalt()	Reaching the target speed of 0 (interpolator stop).

Tab. 1: Overview of Set Change Conditions

MCX axes can process the buffer in two ways:

– *Normal*

  The buffer is processed once. After that, the system waits for new movement commands.

– *Cyclic*

– Processing of the buffer restarts after each pass with the first movement command and does not end until it is explicitly terminated (Delete()).

3.1 Syntax for MCAxis.Buffer

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Buffer.<Function>(optional <Parameter>);
```

3.2 Functions of MCAxis.Buffer

This function offers the following functions:

– Create()
– Clear()
– Delete()
– Suspend()
– Resume()
– Wait()

3.2.1 Create()

This function enables the buffer for this axis. This changes the axis's operating state from *normal* to *buffered*.

transfer parameter

Parameters	data type	Meaning
MCBufferTypes	MCBufferTypes (optional)	Defines the buffer type.
	MCBufferTypes.Normal (Active after reset)	Defines a buffer whose contents are processed once.
	MCBufferTypes.Cyclic	Defines a buffer whose contents are processed from the beginning until processing is explicitly terminated.

Syntax

```
AX1.Buffer.Create();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before issuing any further commands for the axis, check whether the new operating state has been reached in the program as follows:

```
When AX1.Buffer.BufferState == MCBufferStates.Normal, continue;
```

or

```
When AX1.Buffer.BufferState == MCBufferStates.Cyclic , continue;
```

3.2.2 Clear()

This function disables buffer processing, with the following effects:

– The axis operating state changes from *buffered* to *normal*.

– The buffer is cleared.

– A movement that has already begun will be carried through to the end.

The `Power.Disable()` and `Power.Quickstop()` commands also cause the buffer to be disabled.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Buffer.Clear();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

3.2.3 Delete()

This function is relevant only for the cyclic buffer type. It tells the system where the cyclic buffer content ends. In other words, when processing of this command begins, the buffer starts being processed again from the command that was written after `Buffer.Create()`.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Buffer.Delete();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

3.2.4 Suspend()

This function pauses buffer processing at the next block transition, with the following effects:

– The current movement is completed.
– The next movement described in the buffer will not be initiated.
– The axis remains in the *buffered* operating state.

Calling the Resume() [▶ 35] function resumes processing of the buffer.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Buffer.Suspend();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

3.2.5 Resume()

This function resumes processing of the buffer that was paused by Suspend() [▶ 35] .

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Buffer.Resume();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

3.2.6 Wait()

The function sets a delay in the buffer that postpones the execution of a command relative to the command written to the buffer immediately before it.

transfer parameter

Parameters	data type	Meaning
Waiting Time	Integer (optional)	Defines the wait time in [ms].

Syntax

```
AX1.Buffer.Wait(100);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

3.3 Properties of MCAxis.Buffer

This function offers the following properties:

– BufferState
– IsCyclic
– IsNormal

3.3.1 BufferState

Description

This property allows you to read the current buffer status of the MCX axis.

After power-up, the axis is in *normal*operating state, i.e., in unbuffered mode. The buffer status is `0`.

data type	Meaning
MCBufferStates	Buffer status of the MCX axis:
0	The axis is in *normal*operating state: no buffered operation.
Normal (1)	The axis is in *normal*buffer status: The buffer contents are processed once.
Cyclic (2)	The axis is in *cyclic*buffer status: The buffer contents are processed continuously.

Syntax

```
If Not AX1.State.IsSimulation then
...
End_If;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

3.3.2 IsCyclic

Description

This property checks whether the current buffer status of the MCX axis is *cyclic* .

data type	Meaning
Bool	Buffer status of the MCX axis:
True	The axis is in *cyclic*buffer status: The buffer contents are processed continuously.
False	– The axis is not in buffered mode. – The axis is in *normal*buffer status.

Syntax

```
If AX1.Buffer.IsCyclic then
```

state administration

The property can be queried in any state.

3.3.3 IsNormal

Description

This property checks whether the current buffer status of the MCX axis is *normal* .

data type	Meaning
Bool	Buffer status of the MCX axis:
True	The axis is in *normal*buffer status: The buffer contents are processed once.
False	– The axis is not in buffered mode. – The axis is in *cyclic*buffer status.

Syntax

```
If AX1.Buffer.IsNormal then
```

state administration

The property can be queried in any state.

3.4 Example task for MCAxis.Buffer

Here is an example task for a single axis operating in the *"cyclic"* buffer type.

A cyclic buffer is created that starts reversing an axis between positions 1000 and 0 at the speed specified by the `Speed` variable. When it reaches the target, the axis pauses for 200 ms.

```
Task Sample 3 : Autorun

    // Turning on the servo amplifier
    If Not AX1.State.IsEnabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_IF;
    When AX1.State.IsEnabled , continue;

    // Enable cyclic buffer mode
    AX1.Buffer.Create(MCBufferTypes.Cyclic);
    When AX1.Buffer.IsCyclic Continue;

    // Fill the buffer with commands for reverse operation
    AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,1000,
Speed,1000,5000,0.1);
    AX1.Buffer.Wait(200);
    AX1.MovePtp.Start(,0,,,,);
    AX1.Buffer.Wait();
    AX1.Buffer.Delete(); // Indicates the end of the circular Buffer to the
MCX

    // Stop reversing after 10 seconds
    delay(t#10s);
    AX1.Buffer.Suspend();

    // Resume after another 5 seconds
    delay(t#5s);
AX1.Buffer.Resume();
```

```
    // After another 20 seconds, stop the reverse operation and
    // Lock the axis when the last movement is complete
    delay(t#20s);
AX1.Buffer.Clear();
    When (AX1.Buffer.BufferState == 0) AND (AX1.State.IsRunning is false)
Continue;

    AX1.Power.Disable();
    When AX1.State.IsDisabled , continue;

End_Task;
```

4 MCAxis.Diagnostics

The **MCAxis.Diagnostics** function allows you to manage error handling for an MCX axis.

ⓘ Info

Scope of the function

With the exception of the `ClearErrors()` function, the functions and properties listed here refer to MCX errors. Errors and warnings detected by the servo amplifiers (e.g., JetMove 3xxx) must be handled using the **MCAxis.Drive.Diagnostics** function.

The Motion Control eXtended system distinguishes between *errors* and *messages*. Errors trigger a system response, whereas messages do not. Errors and messages are logged in separate buffers, from which they can be read. The buffer collects the events that have occurred since the last call to the `ClearErrors`() function (see ClearErrors() [▶ 40]).

The meanings of the error and message numbers are listed in the JetSym online help under “MotionControl Errors.”

4.1 Syntax for MCAxis.Diagnostics

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Diagnostics.<Function or Access to Value>(optional <Parameter>);
```

4.2 Features of MCAxis.Diagnostics

This function offers the following functions:

– ClearErrors()
– GetErrorCode(BufferIndex)
– GetMessageCode(BufferIndex)

4.2.1 ClearErrors()

If an error occurs, the function acknowledges the error condition. The entries in the error buffer are deleted.

Messages do not trigger an error response and therefore do not need to be acknowledged. For messages, this function clears the entries in the message buffer.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Diagnostics.ClearErrors();
```

ⓘ Info

Scope of the order

This command clears all errors associated with the axis. These can be errors reported by the servo amplifier, as well as errors in the motion control section of the controller (MCX).

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state	
Blocked	–	Blocked	
Approved	–	Approved	
Working	–	Working	
Interrupted	–	Interrupted	
Error: Approved	√	Approved	The system will only switch to this state once the cause of the error has been resolved. Otherwise, the error condition may persist or recur immediately.
Error: blocked	√	Blocked	

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

4.2.2 GetErrorCode()

This function returns the error number at the error buffer location specified by `BufferIndex` . If no `BufferIndex` is provided, the number of the most recently recorded error is returned (`BufferIndex` = `0`).

transfer parameter

Parameters	data type	Meaning
BufferIndex	Integer (optional)	The `BufferIndex` parameter specifies the position in the error buffer from which the error number is to be read. The maximum value of the `BufferIndex` parameter can be retrieved via the `ErrorCount` property.

Syntax

```
AX1.Diagnostics.GetErrorCode();
```

state administration

The function can be read in any state.

4.2.3 GetMessageCode()

This function returns the message number at the message buffer location specified by `BufferIndex` . If no `BufferIndex` is provided, the number of the most recently recorded message is returned (`BufferIndex` = 0).

transfer parameter

Parameters	data type	Meaning
BufferIndex	Integer (optional)	The `BufferIndex` specifies the position in the message buffer from which the message number is to be read. The maximum value of the `BufferIndex` parameter can be retrieved via the `MessageCount` property.

Syntax

```
AX1.Diagnostics.GetMessageCode();
```

state administration

The function can be read in any state.

4.3 Properties of MCAxis.Diagnostics

This function offers the following properties:

- ErrorCount
- IsErrorPending
- MessageCount
- IsMessagePending

4.3.1 ErrorCount

Description

property	data type	Meaning
ErrorCount	Integer	Reads the number of errors logged since the last time the `ClearErrors()` function was called or since the controller was restarted.

Syntax

```
ErrorCount := AX1.Diagnostics.ErrorCount;
```

state administration

The property can be read in any state.

4.3.2 IsErrorPending

Description

data type	Meaning
Bool	Check whether there is an error in the MCX system. This is equivalent to comparing `ErrorCount` to 0.

Syntax

```
If AX1.Diagnostics.IsErrorPending then
...
End_if;
```

state administration

The property can be queried in any state.

4.3.3 IsMessagePending

Description

data type	Meaning
Bool	Check whether there is a message in the MCX system. This is equivalent to comparing `MessageCount` to `0`.

Syntax

```
If AX1.Diagnostics.IsMessagePending then
...
End_if;
```

state administration

The property can be queried in any state.

4.3.4 MessageCount

Description

data type	Meaning
Integer	Reads the number of logged messages since the last time the `ClearErrors()` function was executed or since the controller was restarted.

Syntax

```
MessageCount := AX1.Diagnostics.MessageCount;
```

state administration

The property can be read in any state.

4.4 Example task for MCAxis.Diagnostics

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    Axis Error: Array[10] of INT; //Variables for displaying the error
buffer
    i: INT; //Loop variable
    AcknolegeError: Bool; // Auxiliary flag for an acknowledgment condition

End_Var;

Task Sample_Error Autorun

Loop

    //Clear error variables for display
    For i := 0 To 9 Do
        AchsError[i] := 0;
    End_For;
    AcknowledgeError := False;

    If AX1.Diagnostics.IsErrorPending is true, continue;
```

```
        Taskbreak Automatic;
        // Add additional error handling here

        //Store errors in variables, e.g., for display purposes
        For i := 0 To AX1.Diagnostics.ErrorCount- 1 Do
            AchsError[i] := AX1.Diagnostics.GetErrorCode(i);
        End_For;

    // Bugs have been fixed
    When AcknowledgeError , continue;

        AX1.Diagnostics.ClearErrors();
        Automatictask restart ;
End_Loop;

End_Task;
```

5 MCAxis.Drive.Amplifier

The **MCAxis.Drive.Amplifier** function allows you to read and adjust the properties of the MCX axis's servo amplifier.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the "Amplifier" commissioning page for Bucher-Automation axes Bucher-Automation. When the controller is initial-ized, the parameters are set to the values saved during commissioning.

The Motion API functions allow you to change or read these parameters via the STX program as well.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for down-load on the Bucher Automation website.

5.1 Syntax for MCAxis.Drive.Amplifier

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Amplifier.<Access to Value>;
```

5.2 Properties of MCAxis.Drive.Amplifier

This function offers the following properties:

- ActualDCLinkVoltage
- ActualHeatSinkTemperature
- ActualInternalTemperature
- DCLinkVoltageMax
- DCLinkVoltageMin
- PwmFrequency
- PwmFrequencyReduction
- RatedCurrent
- RatedDCLinkVoltage
- WireTestMode
- I2tValue

5.2.1 ActualDCLinkVoltage

Description

data type	Meaning
Double (optional)	Reads the current DC link voltage of the servo amplifier.

Syntax

```
DCLinkVoltage := AX1.Drive.Amplifier.ActualDCLinkVoltage;
```

state administration

The properties can only be accessed in read-only mode, since the DC link voltage is determined solely by the hardware and the mechanical load.

The property can be read in any state.

5.2.2 ActualHeatSinkTemperature

Description

data type	Meaning
Double (optional)	Reads the current heat sink temperature of the servo amplifier.

Syntax

```
HeatSinkTemperature := AX1.Drive.Amplifier.ActualHeatSinkTemperature;
```

state administration

These properties are read-only, since the heat sink temperature is determined solely by the hardware and the load current.

The property can be read in any state.

5.2.3 ActualInternalTemperature

Description

data type	Meaning
Double (optional)	Reads the actual internal temperature of the servo amplifier.

Syntax

```
InteriorTemperature := AX1.Drive.Amplifier.ActualInternalTemperature;
```

state administration

These properties are read-only, as the interior temperature is determined solely by the hardware and the ambient temperature.

The property can be read in any state.

5.2.4 DCLinkVoltageMax

Description

The DC link voltage is determined by the input voltage and the energy recovered during braking. If it exceeds the value defined here, an overvoltage fault is triggered, which prevents the voltage from rising further.

data type	Meaning
Double (optional)	Reads the fault trigger threshold for overvoltage in the servo amplifier's DC link in [V_{dc}].

Syntax

```
MaxIntermediateCircuitVoltage := AX1.Drive.Amplifier.DCLinkVoltageMax;
```

state administration

This property is read-only because this error threshold is determined by the hardware to protect the device.

The property can be read in any state.

5.2.5 DCLinkVoltageMin

Description

If the DC link voltage drops too low due to insufficient mains voltage or an excessive load, a fault is triggered. This property allows you to read the error threshold. For some device types, it is also possible to adjust the error threshold; however, this is recommended only in very rare cases.

data type	Meaning
Double (optional)	Reads or writes the fault trigger threshold for undervoltage in the servo amplifier's DC link in [V_{dc}].

Syntax

```
MinDCLinkVoltage := AX1.Drive.Amplifier.DCLinkVoltageMax;
AX1.Drive.Amplifier.DCLinkVoltageMax := 12.0;
```

state administration

The property can be read and written in any state.

5.2.6 PwmFrequency

Description

As a rule, the default value is the best choice, since the servo amplifier is optimized for this frequency. The power dissipation of the servo amplifier changes depending on the setting of this property.

In a few cases, it may be advisable to change this value:

– Under very high loads and at low speeds, a lower value may be appropriate.

– At very high speeds and/or during extremely dynamic positioning, a higher value may be appropriate.

This property cannot be changed on all devices.

data type	Meaning
Drive PWM Frequencies (optional)	Reads or writes the inverter switching frequency in the servo amplifier (PWM frequency) in [kHz]. Frequency options as specified in the data type.
DrivePwmFrequencies.kHz4	
DrivePwmFrequencies.kHz8	
DrivePwmFrequencies.kHz16	

Syntax

```
PWMFrequency := AX1.Drive.Amplifier.PwmFrequency;
AX1.Drive.Amplifier.PwmFrequency := DrivePwmFrequencies.kHz8;
```

state administration

The property can be read and written in any state.

5.2.7 PWMFrequencyReduction

Description

On some devices, this property can be enabled to automatically reduce the PWM frequency at very low motor speeds. In exceptional cases, this may be useful for reducing the power dissipation of the servo amplifier under high load and at very low speeds.

This property cannot be changed on all devices.

data type	Meaning
Drive PWM Frequency Reduction Modes (optional)	Reads or writes the reduction in the inverter's PWM frequency in the servo amplifier. Frequency options as specified.
DrivePwmFrequenciesReductionModes.Disable	Disables the property.
DrivePwmFrequenciesReductionModes.Enable	Enables the property.

Syntax

```
PWMFrequencyReduction := AX1.Drive.Amplifier.PwmFrequencyReduction;
AX1.Drive.Amplifier.PwmFrequencyReduction := DrivePwmFrequencyReduction-
Modes.Disabled;
```

state administration

The property can be read and written in any state.

5.2.8 RatedCurrent

Description

data type	Meaning
Double (optional)	Reads the rated current of the servo amplifier assigned to this axis in [A_{ac}].

Syntax

```
DeviceRatedCurrent := AX1.Drive.Amplifier.RatedCurrent;
```

state administration

The property can be read in any state.

5.2.9 RatedDCLinkVoltage

Description

data type	Meaning
Double (optional)	Reads the nominal DC link voltage of the servo amplifier assigned to this axis in [V_{dc}].

Syntax

```
DeviceRatedDCLinkVoltage := AX1.Drive.Amplifier.RatedDCLinkVoltage;
```

state administration

The property can be read in any state.

5.2.10 WireTestMode

Description

It may take a short time to process this property. If it cannot be used in the application for that reason, the feature can be configured.

Not all devices support the motor cable test.

data type	Meaning
DriveWireTestModes (optional)	Indicates whether the servo amplifier performs a motor cable test when the motor function is enabled for the first time.
DriveWireTestModes.Disable	Disables the property.
DriveWireTestModes.Enable	Enables the property.

Syntax

```
MotorCableTestStatus := AX1.Drive.Amplifier.WireTestMode;
AX1.Drive.Amplifier.WireTestMode := DriveWireTestModes.Disabled;
```

state administration

The property can be read and written in any state.

5.2.11 I2tValue

Description

Not all devices support the calculation of the I²t value for the servo amplifier.

data type	Meaning
Double (optional)	Reads the integral value of the I²t monitoring for the servo amplifier assigned to this axis, expressed as a percentage (%)

Syntax

```
DeviceI2tValue := AX1.Drive.Amplifier.I2tValue;
```

state administration

The property can be read in any state.

6 MCAxis.Drive.Brake

The **MCAxis.Drive.Brake** function allows you to read and adjust the properties of the MCX axis's motor holding brake.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the "Brake" commissioning page for Bucher-Automation axes. When the controller is initialized, the parameters are set to the values saved during commissioning.

The Motion API functions allow you to change or read these parameters via the STX program as well.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

6.1 Syntax for MCAxis.Drive.Brake

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Brake.<Function or Access to Value>;
```

6.2 Features of MCAxis.Drive.Brake

This function offers the following functions:

– Apply()
– Release()

6.2.1 Apply()

Description

This function engages the motor holding brake. This only works if the `EncDriv.Drive.Brake.Mode` property is set to " *manual* ". In the *auto* operating state, the servo amplifier controls the brake based on the drive status.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Drive.Brake.Apply();
```

state administration

The function can be called from any state.

6.2.2 Release()

Description

This function releases the motor holding brake. This only works if the `EncDriv.Drive.Brake.Mode` property is set to *"manual"*. In the *auto* operating state, the servo amplifier controls the brake based on the drive status.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Drive.Brake.Release();
```

state administration

The function can be called from any state.

6.3 Properties of MCAxis.Drive.Brake

This function offers the following properties:

– ApplyDelay
– ReleaseDelay
– Mode
– State

6.3.1 ApplyDelay

Description

This delay causes the brake output to be activated first when the axis is *disabled*, and the drive is de-energized only after the delay time set here has elapsed. This prevents, for example, a vertical axis from sagging when the brakes are applied, even though the brakes cannot take effect immediately due to the mechanical movement of the brake pads.

data type	Meaning
Double	Reads or writes the delay time when releasing the motor holding brake in [ms]. The delay only takes effect if the `EncDriv.Drive.Brake.Mode` property is set to *"auto"*.

Syntax

```
BrakeDelayClose := AX1.Drive.Brake.ApplyDelay;
AX1.Drive.Brake.ApplyDelay := 150.0;
```

state administration

The property can be read and written in any state.

6.3.2 ReleaseDelay

Description

This delay causes the brake output to be activated first when the axis is *enabled*, and the `isEnabled` signal becomes active only after the delay time set here has elapsed. The output for the motor holding brake activates immediately when power is applied to the motor. Since the brake may not release immediately, the delay ensures that the motor does not move until the brake has fully released.

data type	Meaning
Double	Reads or writes the delay time when releasing the motor holding brake in [ms]. The delay only takes effect if the `EncDriv.Drive.Brake.Mode` property is set to *"auto"*.

Syntax

```
BrakeReleaseDelay := AX1.Drive.Brake.ReleaseDelay;
AX1.Drive.Brake.ReleaseDelay := 30.0;
```

state administration

The property can be read and written in any state.

6.3.3 Mode

Description

This property allows you to read or adjust the operating state for the motor holding brake.

data type	Meaning
DriveBrakeModes	Operating state of the motor holding brake:
DriveBrakeModes.Manual	The motor holding brake is not controlled by the servo amplifier. The user controls the brake directly through the program or the motion setup.
DriveBrakeModes.Auto	Select this operating state in the following cases: – No brake is connected. – The brake should be applied for maintenance purposes when the motor is de-energized.

Syntax

```
BrakeMode := AX1.Drive.Brake.Mode;
AX1.Drive.Brake.Mode := DriveBrakeModes.Manual;
```

state administration

The property can be read and written in any state.

6.3.4 State

Description

This property allows you to read the status of the motor holding brake.

data type	Meaning
DriveBrakeStates	Position of the motor holding brake:
DriveBrakeStates.Applied	The motor holding brake is engaged.
DriveBrakeStates.Released	The motor holding brake is disengaged.

Syntax

```
BrakeState := AX1.Drive.Brake.State;
```

state administration

The property can be read in any state.

7 MCAxis.Drive.Control.Current

The **MCAxis.Drive.Control.Current** function allows you to read and adjust the properties of the current/ torque controller for the MCX axis's servo amplifier.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the commissioning pages "Control > Current/Torque " and Limit Values for Bucher-Automation Axes." When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read these parameters via the STX program and adjust some of them.

In a servo amplifier, the motor control is designed so that the motor current is proportional to the motor torque. For this reason, such a controller can be referred to as both a current controller and a torque controller, even though in reality only the current is controlled. The torque values are calculated by multiplying the current values by the torque constant.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

7.1 Syntax for MCAxis.Drive.Control.Current

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Control.Current.<Access to Value>;
```

7.2 Properties of MCAxis.Drive.Control.Current

This function offers the following properties:

- Kp
- Tn
- ActualMotorCurrent
- ActualMotorVoltage
- MaximumMotorCurrent

7.2.1 Kp

Description

This property allows you to read or adjust the gain value for the servo amplifier's current controller.

The current controller is designed as a PI controller. The setting of the current controller depends largely on the motor specifications and can therefore be easily calculated. This is the recommended procedure during commissioning. The current controller can be automatically tuned in JetSym 's motion setup using the Controller Optimization button.

data type	Meaning
Double	Defines the gain of the current controller in [V/A].

Syntax

```
CurrentControllerGain := AX1.Drive.Control.Current.Kp;
AX4.Drive.Control.Current.Kp := 1.45;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode. In most cases, however, this is only useful during the initial commissioning of the axis for the purpose of tuning the control system. **NOTICE! Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.**

7.2.2 Tn

Description

This property allows you to read or adjust the settling time for the servo amplifier's current controller.

The current controller is designed as a PI controller. The setting of the current controller depends largely on the motor specifications and can therefore be easily calculated. This is the preferred procedure during commissioning. The current controller can be automatically tuned in JetSym 's motion setup using the Controller Optimization button.

data type	Meaning
Double	Defines the current controller's settling time in [ms].

Syntax

```
CurrentControllerResponseTime := AX1.Drive.Control.Current.Tn;
AX4.Drive.Control.Current.Tn := 2.5;
```

state administration

The property can be read in any state.

This property can also be accessed in a write operation. In most cases, however, this is only useful during the initial commissioning of the axis for the purpose of tuning the control system. **NOTICE! Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.**

7.2.3 ActualMotorCurrent

Description

data type	Meaning
Double	Reads the current motor phase current output by the servo amplifier in [A_{eff}].

Syntax

```
IstStrom := AX1.Drive.Control.Current.ActualMotorCurrent;
```

state administration

The property can be read in any state.

7.2.4 ActualMotorVoltage

Description

data type	Meaning
Double	Reads the current motor voltage between two motor phases, as output by the servo amplifier, in [Vrms].

Syntax

```
IsVoltage := AX1.Drive.Control.Current.ActualMotorVoltage;
```

state administration

The property can be read in any state.

7.2.5 MaximumMotorCurrent

Description

data type	Meaning
Double	Reads the maximum available motor phase current, which is provided by the servo amplifier based on the settings for the motor rated torque, the overload torque factor, and the torque constant, in [A_{eff}].

Syntax

```
MaxCurrent := AX1.Drive.Control.Current.MaxMotorCurrent;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

- ► Avoid making changes while the machine is running.
- ► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

8 MCAxis.Drive.Control.Position

The **MCAxis.Drive.Control.Position** function allows you to read and adjust the properties of the position controller for the MCX axis's servo amplifier.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the commissioning pages for Bucher-Automation axes. When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read these parameters via the STX program and adjust some of them.

The position controller in a servo amplifier enables precise control of the motor's position based on the set-point generated by the controller. It is designed as a P-controller with speed feedforward.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

8.1 Syntax for MCAxis.Drive.Control.Position

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Control.Position.<Function or Access to Value>;
```

8.2 Functions of MCAxis.Drive.Control.Position

This function offers the following functions:

– ResetTrailingIndicators()

8.2.1 ResetTrailingIndicators()

Description

The tracking error measurement records the maximum position tracking error that has occurred in both the positive and negative directions. The `ResetTrailingIndicators()` function resets the trailing indicator count to zero and starts a new measurement.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.Drive.Control.Position.ResetTrailingIndicators();
```

state administration

The function can be called from any state.

8.3 Properties of MCAxis.Drive.Control.Position

This function offers the following properties:

– Kv
– FeedbackSource
– SpeedFeedForwardScale

8.3.1 Kv

Description

The position controller is designed as a P-controller with speed feedforward. The higher the gain value, the more dynamic the drive becomes. If the values are too high, the drive will start to vibrate.

data type	Meaning
Double	Reads or changes the gain of the position controller, in [1/s].

Syntax

```
PositionControllerGain := AX1.Drive.Control.Position.Kv;
AX4.Drive.Control.Position.Kv := 80.0;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

- ► Avoid making changes while the machine is running.
- ► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

8.3.2 FeedbackSource

Description

In most applications, the first position sensor which is also used for commutation and speed control is calibrated.

In certain cases, the second position sensor can be selected for position control to compensate for slippage between the material and the motor. The second position sensor is then mounted directly on the load, in addition to the motor position sensor. When this additional position sensor is used for position control, the control system automatically compensates for a small deviation between the position of the load and the motor shaft (slip).

data type	Meaning
Drive Feedback Sources	Reads which position sensor has been selected for position control or changes the selection:
DriveFeedbackSources.Encoder1	The position sensor connected to the first encoder interface is used.
DriveFeedbackSources.Encoder2	The position sensor connected to the second encoder interface is being used.

Syntax

```
PositionSensor1 := AX1.Drive.Control.Position.FeedbackSource;
AX1.Drive.Control.Position.FeedbackSource := DriveFeedbackSources.Encoder1;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements when replacing the position sensor during operation

This setting should be configured via the motion setup during the initial commissioning of the axis. If it is necessary to switch between the two position sensors while the machine is running, position jumps may occur, which in turn can lead to uncontrolled movements of the drive. In addition, the reference position may change when the position sensor is replaced.

- ► This property may only be accessed in the **locked** state.

8.3.3 SpeedFeedForwardScale

Description

This property allows you to adjust the effect of speed control when regulating the MCX axis. The speed control compensates for the residual error of the position controller (P controller).

The speed feedforward control determines a target speed for the speed controller by multiplying the calculated target speed by the scaling factor (SpeedFeedForwardScale). This value is added to the position controller's output. A value other than 100% is recommended only under particularly difficult control conditions.

data type	Meaning
Double	Reads or changes the scaling of the speed control effect in [%].

Syntax

```
TorqueFeedforwardScale := AX1.Drive.Control.Position.SpeedFeedForward-
Scale;
AX4.Drive.Control.Position.SpeedFeedForwardScale := 95.0;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

- ► Avoid making changes while the machine is running.
- ► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

9 MCAxis.Drive.Control.Speed

The **MCAxis.Drive.Control.Speed** function allows you to read and adjust the speed controller properties of the MCX axis servo amplifier.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the commissioning pages for Bucher-Automation axes. When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read these parameters via the STX program and adjust some of them.

The speed controller in a servo amplifier allows the motor speed to be dynamically adjusted to meet the requirements of the position controller. It is designed as a PI controller with torque feedforward.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

9.1 Syntax for MCAxis.Drive.Control.Speed

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Control.Speed.<Access to Value>;
```

9.2 Properties of MCAxis.Drive.Control.Speed

This function offers the following properties:

- Kp
- Tn
- Tf
- ActualValue
- Setpoint
- Isum
- IsumPreset
- Overload Factor
- MaxMotorSpeed
- SetpointLimit
- DifferenceWindow
- TotalInertia
- TorqueFeedforwardScale

9.2.1 Kp

Description

This property allows you to read or adjust the gain value for the speed controller of the servo amplifier.

The speed controller is designed as a PI controller. The higher the gain value, the more dynamic the drive becomes. If the values are too high, the drive begins to vibrate.

data type	Meaning
Double	Defines the gain of the speed controller in [Nm/1000 (rpm)].

Syntax

```
SpeedControllerGain := AX1.Drive.Control.Speed.Kp;
AX4.Drive.Control.Speed.Kp := 5.5;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

- ► Avoid making changes while the machine is running.
- ► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

9.2.2 Tn

Description

This property allows you to read or adjust the settling time for the servo amplifier's speed controller.

The speed controller is designed as a PI controller. The lower the value for the settling time, the more responsive the drive becomes. If the values are too low, the drive will start to vibrate. When there are high mechanical inertia masses on the motor shaft, the adjustment time should generally be set longer.

data type	Meaning
Double	Defines the speed controller's settling time in [ms].

Syntax

```
SpeedControllerResponseTime := AX1.Drive.Control.Speed.Tn;
AX4.Drive.Control.Speed.Tn := 10.5;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

- ► Avoid making changes while the machine is running.
- ► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

9.2.3 Tf

Description

This property allows you to read or adjust the filter time constant for the servo amplifier's speed measure-ment.

Since the accuracy of speed measurement is affected by quantization (position differentiation) and by inter-ference on the encoder cable, the actual speed value is passed through a filter.

The following dependencies apply:

– The higher the filter time constant, the smoother the actual speed value becomes.
– The higher the resolution of the position sensor, the lower the filter time constant can be set.
– Since the speed filter introduces a dead time in the speed control, the lower the filter time constant, the higher the gain can be set.

data type	Meaning
Double	Defines the filter time constant for speed detection in [ms].

Syntax

```
FilterTimeConstant := AX1.Drive.Control.Speed.Tf;
AX4.Drive.Control.Speed.Tf := 1.0;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

► Avoid making changes while the machine is running.
► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

9.2.4 ActualValue

Description

data type	Meaning
Double	Reads the current motor speed, which is determined by the servo amplifier's first position sensor, in [rpm].

Syntax

```
Speed := AX1.Drive.Control.Speed.ActualValue;
```

state administration

The property can be read in any state.

9.2.5 Setpoint

Description

data type	Meaning
Double	Reads the current target speed for the speed controller in [rpm]. This is the output value of the position controller; it is limited by the speed limit value.

Syntax

```
TargetSpeed := AX1.Drive.Control.Speed.Setpoint;
```

state administration

The property can be read in any state.

9.2.6 Isum

Description

data type	Meaning
Double	Reads the calculated I-component of the speed controller or writes an application-specific value in [Nm]. The speed controller recalculates this value in each control cycle by adding the current control error.

Syntax

```
CurrentIValue := AX1.Drive.Control.Speed.Isum;
AX4.Drive.Control.Speed.Isum := 0.0;
```

state administration

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

Making a change while the machine is running can cause uncontrolled movements if the entered value does not match the load on the drive.

- ► Avoid making changes while the machine is running.
- ► Changes are generally only advisable during the axis commissioning phase to fine-tune the control system.

One possible use case is as follows: If an axis is intentionally moved into a jammed position (e.g., for pressing), it may be helpful to clear the I component once before initiating positioning to exit the jam.

9.2.7 IsumPreset

Description

This property allows you to read or adjust the default value of the I-component of the PI speed controller when the servo amplifier is enabled.

This value acts as an "initial torque" during pulse release. A non-zero value can be useful for vertical axes to prevent "sagging" when the system is turned on. A suitable value for `IsumPreset` can be determined, for example, by reading the `Isum` value while a vertical axis is at a standstill before locking it.

data type	Meaning
Double	Reads or writes the torque setpoint value for pulse enable [Nm].

Syntax

```
SpeedControllerPreset := AX1.Drive.Control.Speed.IsumPreset;
AX4.Drive.Control.Speed.IsumPreset := 1.25;
```

state administration

The property can be read and written in any state.

9.2.8 OverloadFactor

Description

This property allows you to read or adjust the overload factor for the servo amplifier's speed setting.

Based on the motor's rated speed, this property can be used to determine the maximum permissible speed. Depending on the type of application, this may be higher or lower than the rated value.

data type	Meaning
Double	Reads or writes the speed overload factor [%].

Syntax

```
SpeedOverloadFactor := AX1.Drive.Control.Speed.OverloadFactor;
AX4.Drive.Control.Speed.OverloadFactor := 100.0;
```

state administration

The property can be read and written in any state.

However, a changed value may not take effect until the state is changed (again) to **"Approved** ."

9.2.9 MaxMotorSpeed

Description

data type	Meaning
Double	Reads the maximum available engine speed in [rpm]. This value is calculated based on the motor's rated speed and the overload factor.

Syntax

```
MaxSpeed := AX1.Drive.Control.Speed.MaxMotorSpeed;
```

state administration

The property can be read in any state.

9.2.10 SetpointLimit

Description

This property allows you to set a dynamic speed limit during the machine process. The value is based on the maximum available motor speed (MaxMotorSpeed [▶ 63]).

data type	Meaning
Double	Reads or writes the target speed limit in [%].

Syntax

```
SpeedLimit := AX1.Drive.Control.Speed.SetpointLimit;
AX4.Drive.Control.Speed.SetpointLimit := 100.0;
```

state administration

The property can be read and written in any state.

9.2.11 DifferenceWindow

Description

This property allows you to view or adjust the threshold for monitoring speed deviation. If the drive cannot follow the position controller's command and a deviation between the setpoint and actual speed occurs that exceeds the value defined in this property, an error is triggered. This method is used, for example, to monitor for unintended jamming of a drive.

data type	Meaning
Double	Reads or writes the error threshold for the deviation between the actual and target speed in [%].

Syntax

```
SpeedErrorLimit := AX1.Drive.Control.Speed.DifferenceWindow;
AX4.Drive.Control.Speed.DifferenceWindow := 10.0;
```

state administration

The property can be read and written in any state.

9.2.12 TotalInertia

Description

This property allows you to view or adjust the setting for the mechanical system's moment of inertia relative to the motor shaft. The value of this property is used as a factor in the calculation of torque control. Torque feedforward has a significant impact on the control performance of an MCX axis, particularly in highly dynamic applications.

The torque feedforward calculates a target torque for the torque controller based on the calculated target acceleration multiplied by the scaling factor (Torque Feedforward Scale [▶ 65]) and the total inertia (`TotalInertia`). This value is added to the speed controller's output.

data type	Meaning
Double	Reads or writes the total moment of inertia relative to the motor shaft in [kg cm²].

Syntax

```
TotalInertia := AX1.Drive.Control.Speed.TotalInertia;
AX4.Drive.Control.Speed.TotalInertia := 8.53;
```

state administration

The property can be read and written in any state.

9.2.13 TorqueFeedforwardScale

Description

This property allows you to view or adjust the scaling of the torque control effect. The value of this property is used as a factor in the calculation of torque control. Torque feedforward has a significant impact on the control performance of an MCX axis, particularly in highly dynamic applications.

The torque feedforward calculates a target torque for the torque controller based on the calculated target acceleration multiplied by the scaling factor (`TorqueFeedForwardScale` and the total inertia (TotalInertia [▶ 64]. This value is added to the speed controller's output.

data type	Meaning
Double	Reads or writes the scaling factor for the torque control effect in [%].

Syntax

```
TorqueFeedForwardScale := AX1.Drive.Control.Speed.TorqueFeedForwardScale;
AX4.Drive.Control.Speed.TorqueFeedForwardScale := 90.0;
```

state administration

The property can be read and written in any state.

10 MCAxis.Drive.Control.Torque

The **MCAxis.Drive.Control.Torque** function allows you to read and adjust the properties of the current/ torque controller in the MCX axis's servo amplifier.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the commissioning pages for Bucher-Automation Axes." When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read these parameters via the STX program and adjust some of them.

In a servo amplifier, the motor control is designed so that the motor current is proportional to the motor torque. For this reason, such a controller can be referred to as both a current controller and a torque controller, even though in reality only the current is controlled. The torque values are calculated by multiplying the current values by the torque constant.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

10.1 Syntax for MCAxis.Drive.Control.Current

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Control.Torque.<Access to Value>;
```

10.2 Properties of MCAxis.Drive.Control.Torque

This function offers the following properties:

- ActualValue
- Setpoint
- Overload Factor
- MaxMotorTorque
- SetpointLimit

10.2.1 ActualValue

Description

data type	Meaning
Double	Reads the current motor torque, which is calculated from the measured motor current of the servo amplifier, in [Nm]

Syntax

```
IsTorque := AX1.Drive.Control.Torque.ActualValue;
```

state administration

The property can be read in any state.

10.2.2 Setpoint

Description

data type	Meaning
Double	Reads the current target torque for the torque/current controller in [Nm]. This is the output value of the speed controller; it is limited by the speed limit value.

Syntax

```
TargetTorque := AX1.Drive.Control.Torque.Setpoint;
```

state administration

The property can be read in any state.

10.2.3 OverloadFactor

Description

Based on the motor's rated torque, this property can be used to determine the maximum permissible torque. Depending on the type of application, this may be higher or lower than the rated value.

data type	Meaning
Double	Reads or writes the overload factor for the servo amplifier's torque setting in [%].

Syntax

```
OverloadFactor := AX1.Drive.Control.Torque.OverloadFactor;
AX4.Drive.Control.Torque.OverloadFactor := 200.0;
```

state administration

The property can be read and written in any state.

However, a changed value may not take effect until the state is changed (again) to **"Released** ."

10.2.4 MaxMotorTorque

Description

data type	Meaning
Double	Reads the maximum available engine torque in [Nm]. This value is calculated based on the motor's rated torque and the overload factor.

Syntax

```
MaxTorque := AX1.Drive.Control.Torque.Torque;
```

state administration

The property can be read in any state.

10.2.5 SetpointLimit

Description

This property allows you to set a dynamic torque limit during the machine process. This value is based on the maximum available motor torque (MaxMotorTorque [▶ 67].

data type	Meaning
Double	Reads or writes the target speed limit in [%].

Syntax

```
TorqueLimit := AX1.Drive.Control.Torque.SetpointLimit;
AX4.Drive.Control.Torque.SetpointLimit := 25.3;
```

state administration

The property can be read and written in any state.

11 MCAxis.Drive.Diagnostics

The **MCAxis.Drive.Diagnostics** function allows you to manage error handling for the servo amplifier of an MCX axis.

Scope of the function

The functions and properties listed here refer to errors and warnings detected by the servo amplifiers (e.g., JetMove 3xxx).

Errors trigger a system response, whereas warnings do not. Errors must be acknowledged by the user in order to resume operation of the drive, while warnings are automatically cleared once the cause has been resolved.

The meanings of the error codes are listed in the JetSymonline help or in the servo amplifier's user manual.

Use the `MCAxis.Diagnostics.ClearErrors()` command to clear all errors, including those from the servo amplifiers.

Errors and messages related to position generation functions (MCX) are managed via the **MCAxis.Diagnostics** function.

11.1 Syntax for MCAxis.Drive.Diagnostics

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Diagnostics.<Function or Access to Value>;
```

11.2 Features of MCAxis.Drive.Diagnostics

This function offers the following functions:

– Is WarningPending()

11.2.1 IsWarningPending()

The function returns whether a specific warning from the servo amplifier is present, or whether any warning is present at all. Not every type of servo amplifier provides all warnings.

transfer parameter

Parame-ters	data type	Meaning
Warning No.	DriveWarningTypes (optional)	
	DriveWarningTypes.AnyWarning (De-fault)	The servo amplifier displays a warning.
	DriveWarningTypes.UndervoltageD-CLink	– The DC link voltage is too low. – A DC link voltage is present while the control system is disabled.
	DriveWarningTypes.Overtempera-tureDrive	The temperature of the heat sink or the interior of the device is close to the overheating shutdown threshold.
	DriveWarningTypes.Overtemperature Motor	The engine temperature is close to the overheating shutdown threshold.
	DriveWarningTypes.STOactive	The STO function is active while control is disabled.
	DriveWarningTypes.PowerSupply	There is a warning in the power supply.
	DriveWarningTypes.I2TDrive	The I²t value for the device current is close to the turn-off threshold.
	DriveWarningTypes.I2TMotor	The I²t value for the motor current is close to the trip threshold.

Syntax

```
If AX1.Drive.Diagnostics.IsWarningPending(DriveWarningTypes.AnyWarning) then
...
End_if
```

state administration

The function can be called from any state.

11.3 Properties of MCAxis.Drive.Diagnostics

This function offers the following properties:

– ActualError
– IsErrorPending

11.3.1 ActualError

Description

property	data type	Meaning
ActualError	dword	Reads the error code for an error occurring on the servo amplifier. If the return value is `0`, the servo amplifier is fault-free.

Syntax

```
AxisError := AX1.Drive.Diagnostics.ActualError;
```

state administration

The property can be read in any state.

11.3.2 IsErrorPending

Description

data type	Meaning
Bool	Check whether there is an error in the servo amplifier of the MCX axis. `0`: The servo amplifier is working properly. `1`: The servo amplifier is indicating an error. This is equivalent to comparing `ErrorCount` to `0`.

Syntax

```
If AX1.Drive.Diagnostics.IsErrorPending then
...
End_if;
```

state administration

The property can be queried in any state.

12 MCAxis.Drive.DigitalInput

The **MCAxis.Drive.DigitalInput** function allows you to read and adjust the properties of the digital inputs on the MCX axis servo amplifier.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the commissioning page for Bucher-Automation axes. When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read these parameters via the STX program and adjust some of them.

The signals at the digital inputs are used in the MCX for reference switch, limit switch, and STO (safe torque-off functions. You can check the state of the limit and reference switches via the properties of the MCAxis.Mechanism.Monitoring [▶ 132] function.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

12.1 Syntax for MCAxis.Drive.DigitalInput

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.DigitalInput.<Access to Value>;
```

12.2 Properties of MCAxis.Drive.DigitalInput

This function offers the following properties:

- PolarityReferenceSwitch
- LimitSwitchMode
- PolarityLimitSwitchMin
- PolarityLimitSwitchMax
- IsPulseEnabled

12.2.1 PolarityReferenceSwitch

Description

This property allows you to read or adjust the polarity setting for the reference switch connected to the corresponding digital input on the servo amplifier. This is how you define the voltage at the digital input at which the reference switch should be considered active.

data type	Meaning
DrivePolarityModes	Reads or writes the polarity setting type.
DrivePolarityModes.HighActive	The reference switch is active when a 24-volt signal is applied to the digital input.
DrivePolarityModes.LowActive	The reference switch is active when a 0-V signal is present at the digital input.

Syntax

```
ReferenceSwitchPolarity := AX1.Drive.DigitalInput.PolarityReferenceSwitch;
AX1.Drive.DigitalInput.PolarityReferenceSwitch := DrivePolarityModes.HighActive;
```

state administration

The property can be read and written in any state.

12.2.2 Limit Switch Mode

Description

This property allows you to read or adjust the settings of the hardware limit switches connected to the corresponding digital input of the servo amplifier. This is how you specify whether an input signal should be used as a limit switch.

data type	Meaning
DriveLimitSwitchModes	Reads or writes the type of limit switch function.
DriveLimitSwitchModes.Disabled	Hardware limit switches are not used (evaluated).
DriveLimitSwitchModes.Both	Both hardware limit switches are used (evaluated).
DriveLimitSwitchModes.Min	Only the hardware limit switch in the negative travel direction is used (detected).
DriveLimitSwitchModes.Max	Only the hardware limit switch in the positive travel direction is used (detected).

Syntax

```
ActiveLimitSwitches := AX1.Drive.DigitalInput.LimitSwitchMode;
AX1.Drive.DigitalInput.LimitSwitchMode := DriveLimitSwitchModes.Both;
```

state administration

The property can be read and written in any state.

12.2.3 PolarityLimitSwitchMin

Description

This property allows you to read or adjust the polarity setting for the hardware limit switch in the negative travel direction that is connected to the corresponding digital input on the servo amplifier. This is how you specify the voltage at the digital input at which the limit switches should be detected as active.

data type	Meaning
DrivePolarityModes	Reads or writes the polarity setting type.
DrivePolarityModes.HighActive	The hardware limit switch in the negative travel direction is active when a 24-V signal is applied to the digital input.
DrivePolarityModes.LowActive	The hardware limit switch in the negative travel direction is active when a 0-V signal is present at the digital input.

Syntax

```
PolarityLimitSwitchMin := AX1.Drive.DigitalInput.PolarityLimitSwitchMin;
AX1.Drive.DigitalInput.PolarityLimitSwitchMin := DrivePolarityModes.LowActive;
```

state administration

The property can be read and written in any state.

12.2.4 PolarityLimitSwitchMax

Description

This property allows you to read or adjust the polarity setting for the hardware limit switch in the positive travel direction that is connected to the corresponding digital input on the servo amplifier. This is how you define the voltage at the digital input at which the limit switches should be detected as active.

data type	Meaning
DrivePolarityModes	Reads or writes the polarity setting type.
DrivePolarityModes.HighActive	The hardware limit switch in the forward direction is active when a 24-V signal is applied to the digital input.
DrivePolarityModes.LowActive	The hardware limit switch in the forward direction is active when a 0-V signal is present at the digital input.

Syntax

```
PolarityLimitSwitchPos := AX1.Drive.DigitalInput.PolarityLimitSwitchMax;
AX1.Drive.DigitalInput.PolarityLimitSwitchMax := DrivePolarityModes.LowActive;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode.

12.2.5 IsPulseEnabled

Description

This property allows you to query the status of the safe stop (STO) inputs on the servo amplifier.

The query returns `TRUE` if the corresponding digital inputs are configured in such a way that the amplifier output stage is enabled, meaning that voltage pulses can be sent to the motor.

data type	Meaning
Bool	Reads the status of the servo amplifier's STO function: `True`: The engine is operational. `False`: The motor is securely locked to prevent torque generation.

Syntax

```
STOStatus := AX1.Drive.DigitalInput.IsPulseEnabled;
```

state administration

The property can be read in any state.

13 MCAxis.Drive.EncoderChannel

The **MCAxis.Drive.EncoderChannel** function allows you to read and adjust the properties of the encoders connected to the MCX axis.

These are essentially the parameters available in the JetSym programming tool under "motion setup" for Bucher-Automationpage Bucher-Automation axes. When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read these parameters via the STX program and, in some cases, modify them.

Servo amplifiers typically have two interfaces for a position encoder. That is why this function is available twice.

Position feedback is required for commutation as well as for speed and position control. Various types of position sensors can be connected. For information on which types are supported in this application, please refer to the servo amplifier's operating instructions.

The second position sensor can be used for two different functions:

1. As an additional position sensor for the axis to compensate for any slippage between the material and the motor.
   In some applications, it makes sense to install a second position sensor directly on the load in addition to the motor position sensor. In this case, the motor position sensor is used for commutation and speed control, and the additional position sensor is used for position control. As a result, position control is performed directly on the load rather than on the motor's rotor position. Any slippage between the motor shaft and the load is thus automatically compensated for.
   Procedure:
   For the parameter `AX1.Drive.Control.Position.FeedbackSource` , select the value `Drive-FeedbackSources.Encoder2,` or in the JetSym motion setup, on the Control > Position page, set the Encoder for Position Control parameter to Encoder 2.
2. As an external axis in a technology group.
   Procedure:
   In the JetSym motion setup, on the Axis Parameters page, select the option "Use Encoder 2 as an external axis."

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

13.1 Syntax for MCAxis.Drive.EncoderChannel

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.EncoderChannel[1..2].<Access to Value>;
```

13.2 Properties of MCAxis.Drive.EncoderChannel

This function offers the following properties:

– EncType
– EncLines
– Gearmotor

– GearEncoder
– MultiTurnPosition
– SingleTurnPosition
– MaxNoOfAbsRevolutions

13.2.1 EncType

Description

This property allows you to read or adjust the type of position sensor connected to the corresponding digital input on the servo amplifier.

data type	Meaning
Drive ncoderTypes	Reads or writes the type of the connected position sensor.
DriveEncoderTypes.Disabled	No position sensor is connected.
DriveEncoderTypes.Resolver	A resolver is connected.
DriveEncoderTypes.Hiperface	A position encoder with a Hiperface interface is connected.
DriveEncoderTypes.EnDat	A position encoder with an EnDat 2.2 interface is connected.
DriveEncoderTypes.RectangleIncrementalEncoder	An incremental encoder with square-wave signals is connected.
DriveEncoderTypes.SinCosIncrementalEncoder	An incremental encoder with sine wave signals is connected.
DriveEncoderTypes.HiperfaceDSL	A position encoder with a Hiperface DSL interface is connected.
DriveEncoderTypes.LinMot	A LinMot drive is connected.

Syntax

```
PositionSensor1 := AX1.Drive.EncoderChannel[1].EncType;
AX1.Drive.EncoderChannel[1].EncType := DriveEncoderTypes.Resolver;
```

state administration

The property can be read and written in any state.

However, a changed value does not take effect until the state transitions from **"Locked"** to **"Unlocked** ."

13.2.2 EncLines

Description

This property allows you to read or adjust the resolution parameter for the position encoder connected to the corresponding position encoder interface on the servo amplifier. Depending on the type of position sensor, the resolution parameter has a different meaning (see the table below).

data type	Position sensor type	Meaning
Uint	-/-	Reads or writes the resolution parameter, the meaning of which depends on the encoder type:
	Resolver	Number of pole pairs (almost always 1).
	Hiperface	The digital position resolution is transmitted at startup. Read-only access.
	EnDat	The digital position resolution is transmitted at startup. Read-only access.
	Rectangular incremental encoder	Number of lines
	Sine-wave incremental encoder	Number of sine wave periods per revolution.
	Hiperface-DSL	Digital resolution (read-only access).
	LinMot	No entry required.

Syntax

```
EncoderLineCount := AX1.Drive.EncoderChannel[1].EncLines;
AX1.Drive.EncoderChannel[1].EncLines := 2500;
```

state administration

The property can be read and written in any state.

However, a changed value does not take effect until the state transitions from **"Locked"** to **"Unlocked** ."

13.2.3 Gearmotor

Description

This property allows you to read or adjust the gear ratio between the motor shaft and the position encoder connected to the corresponding position encoder interface on the servo amplifier.

This property tells the MCX axis how many revolutions the motor shaft rotates when the position encoder rotates by the number of revolutions specified by the Gear Encoder [▶ 77] property.

data type	Meaning
Int	Reads or writes the number of motor revolutions in [rev].

Syntax

```
MotorGearRatioMotor := AX1.Drive.EncoderChannel[1].GearMotor;
AX1.Drive.EncoderChannel[1].GearMotor := 1;
```

state administration

The property can be read in any state.

This property can also be accessed in code.

13.2.4 GearEncoder

Description

This property allows you to read or adjust the gear ratio between the motor shaft and the position encoder connected to the corresponding position encoder interface on the servo amplifier.

This property tells the MCX axis how many revolutions the position encoder rotates when the motor shaft rotates by the number of revolutions specified by the Gearmotor [▶ 77] property.

data type	Meaning
Int	Reads or writes the number of encoder revolutions in [rev].

Syntax

```
EncoderTranslationEncoder := AX1.Drive.EncoderChannel[1].GearEncoder;
AX1.Drive.EncoderChannel[1].GearEncoder := 3;
```

state administration

The property can be read in any state.

This property can also be accessed in a write operation.

13.2.5 MultiTurnPosition

Description

This property allows you to read the current value of the revolution counter for the position encoder connected to the corresponding position encoder interface on the servo amplifier.

data type	Meaning
Integer	Reads the value of the position encoder's revolution counter in [revolutions]. The value is nonzero only for absolute encoders.

Syntax

```
EncoderRevolutions := AX1.Drive.EncoderChannel[1].MultiTurnPosition;
```

state administration

The property can be read in any state.

13.2.6 SingleTurnPosition

Description

This property allows you to read the actual angular position of the shaft of the position encoder connected to the corresponding position encoder interface on the servo amplifier.

data type	Meaning
Int	Reads the angular position of the encoder shaft in [°].

Syntax

```
EncoderAngle:= AX1.Drive.EncoderChannel[1].SingleTurnPosition;
```

state administration

The property can be read in any state.

13.2.7 MaxNoOfAbsRevolutions

Description

This property allows you to read the number of possible absolute revolutions of the encoder connected to the corresponding encoder interface on the servo amplifier.

The value is read from the absolute encoder via the MCX axis and can be used to identify the position encoder.

data type	Meaning
Int	Reads the number of absolute revolutions provided by the position encoder, in [rev]. The value is nonzero only for absolute encoders.

Syntax

```
EncoderNoOfMultiturn := AX1.Drive.EncoderChannel[1].MaxNoOfAbsRevolutions;
```

state administration

The property can be read in any state.

14 MCAxis.Drive.GenericAccess

The **MCAxis.Drive.GenericAccess** function allows you to access specific parameters of the servo amplifier on an MCX axis.

When operating an axis via the MCX kernel from within JetSym, all necessary servo amplifier parameters are typically standardized and made available in SI units through the motion setup and the Motion API functions.

In certain cases, it may be helpful to access a parameter on a servo amplifier that is not available via the Motion API commands. Typically, the Bucher Automation 's technical support team will provide you with Bucher Automation sample command sets for this purpose.

NOTICE



Risk of unintended behavior of the servo amplifier!

Generic access to a parameter via the **MCAxis.Drive.GenericAccess** command performs the desired function only for servo amplifiers of a specific series! This command is not automatically applicable to other series.

► Do not use this command with servo amplifiers from different series.

Generic access to a servo amplifier is implemented in a Bucher-Automationcontrol system via the CANopen protocol. This applies to communication buses based on both CAN and EtherCAT.

CANopen defines an object address (a 16-bit number expressed in hexadecimal) and a subindex (an 8-bit number) for addressing a parameter.

For the command to work correctly, it is also necessary to use the correct data type for the parameter referenced via *the object* and *subindex* . For this reason, the **MXAxis.Drive.GenericAccess** function provides a separate function for reading and writing data for each data type.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

14.1 Syntax for MCAxis.Drive.GenericAccess

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.GenericAccess.CANopen.<Function or Access to
Value>(optional <Paramter 1, Paramter 2, Parameter n>);
```

14.2 Functions of MCAxis.Drive.GenericAccess

This function offers the following functions:

Read access:

– CANopen.ReadUInt8()
– CANopen.ReadInt8()
– CANopen.ReadUInt16()
– CANopen.ReadInt32()
– CANopen.ReadUInt32()
– CANopen.ReadInt32()

– CANopen.ReadFloat32()
– CANopen.ReadFloat64()
– CANopen.ReadVisibleString()

Write access:

– CANopen.WriteUInt8()
– CANopen. WriteInt8()
– CANopen. WriteUInt16()
– CANopen. WriteInt32()
– CANopen. WriteUInt32()
– CANopen. WriteInt32()
– CANopen. WriteFloat32()
– CANopen. WriteFloat64()

14.2.1 CANopen.ReadUInt8()

Description

This function allows you to read a parameter of data type `UInt8` from the servo amplifier.

transfer parameter

Parameters	data type	Meaning
ObjectIndex	Int	The parameter's object address in the servo amplifier.
SubIndex	Word	Subindex number of the CANopen object.

The same procedure applies to all data types

You access objects with a different data type in the same way. The data type of the variable to which the object content is assigned must match the command.

The available commands are:

– CANopen.ReadInt8()
– CANopen.ReadUInt16()
– CANopen.ReadInt32()
– CANopen.ReadUInt32()
– CANopen.ReadInt32()
– CANopen.ReadFloat32()
– CANopen.ReadFloat64()
– CANopen.ReadVisibleString()

Syntax using .ReadUInt8() as an example

```
test1 := AX1.Drive.GenericAccess.CANopen.ReadUInt8(0x2102,0x16);
```

state administration

The function can be called from any state.

14.2.2 CANopen.WriteUInt8()

Description

This function allows you to write a value to a parameter of data type `UInt8` on the servo amplifier.

transfer parameter

Parameters	data type	Meaning
ObjectIndex	Int	The parameter's object address in the servo amplifier.
SubIndex	Word	Subindex number of the CANopen object.
Value	byte	Value to be written to the object.

The same procedure applies to all data types

In the same way, you can write data to objects of a different data type. The data type of the `Value` parameter must match that of the command.

The available commands are:

– CANopen.WriteUInt8()
– CANopen.WriteInt8()
– CANopen.WriteUInt16()
– CANopen.WriteInt32()
– CANopen.WriteUInt32()
– CANopen.WriteInt32()
– CANopen.WriteFloat32()
– CANopen.WriteFloat64()

Syntax, using .WriteUInt8() as an example

```
AX1.Drive.GenericAccess.CANopen.WriteUInt8(0x2101,0x19,1)
```

state administration

The function can be called from any state.

14.3 Properties of MCAxis.Drive.GenericAccess

This function offers the following properties:

– CANopen OS Version
– CANopen.IsAvailable

14.3.1 CANopen.OSVersion

Description

data type	Meaning
String	Reads the firmware version of the MCX axis servo amplifier. The FW version is read as a string for EtherCAT/CANopen objects.

Syntax

```
FW_Version := AX1.Drive.GenericAccess.CANopen.OSVersion;
```

state administration

The property can be read in any state.

14.3.2 CANopen.IsAvailable

Description

data type	Meaning
String	Checks whether the MCX axis provides a CANopen interface.

Syntax

```
If AX1.Drive.GenericAccess.CANopen.IsAvailable Then
…
End_if;
```

state administration

The property can be read in any state.

15 MCAxis.Drive.InvertDirection

The **MCAxis.Drive.InvertDirection** function allows you to read and adjust the direction of rotation of an MCX axis's motor shaft relative to the positioning direction.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

15.1 Syntax for MCAxis.Drive.InvertDirection

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.InvertDirection;
```

15.2 Properties of MCAxis.Drive.InvertDirection

This function offers the following properties:

– InvertDirection

15.2.1 InvertDirection

Description

This property allows you to read or adjust the direction of rotation of the motor shaft relative to the positioning direction.

In a standard wiring configuration, a positioning command in the positive direction typically causes the motor shaft to rotate clockwise (as viewed by the observer). If the `MCAxis.Drive.InvertDirection` property is enabled, a positioning command in the positive direction will instead cause the motor shaft to rotate counter-clockwise.

data type	Meaning
DriveInvertDirectionModes	Reads or sets whether the rotation direction reversal is enabled or inactive.
DriveInvertDirectionModes.Disabled	Rotation direction reversal is disabled.
DriveInvertDirectionModes.Enabled	Reverse rotation is enabled.

Syntax

```
StateDirectionInverting := AX1.Drive.Drive.InvertDirection;
AX1.Drive.InvertDirection := DriveInvertDirectionModes.Enabled;
```

state administration

The property can be read in any state.

Although the property can be written in any state, it does **not make sense** to do so in the **"released"** and **"running"**states. Depending on the type of servo amplifier, write access in these states may cause the input to be ignored until the next state change, or may cause the servo amplifier to generate an error.

Since this is a machine property, it cannot be changed during operation. Instead, this property is stored along with the other axis data in the controller and is reconfigured for the axis when the controller boots up.

16 MCAxis.Drive.IsReadyForOperation

The **MCAxis.Drive.IsReadyForOperation** function allows you to determine whether the servo amplifier is ready to be switched to the **enabled** state.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

16.1 Syntax for MCAxis.Drive.IsReadyForOperation

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.IsReadyForOperation;
```

16.2 Properties of MCAxis.Drive.IsReadyForOperation

This function offers the following properties:

– Ready for Operation

16.2.1 IsReadyForOperation

Description

It is recommended that you check this property before issuing the `Power.Enable` command. This property is usually `true` if the following conditions are met:

– There is a communication connection to the servo amplifier.
– The servo amplifier is working properly.
– The DC link voltage is sufficiently high (mains power is connected).
– The STO (safe stop) signal is not active.
– A position sensor has been configured.

data type	Meaning
Bool	Checks whether the servo amplifier is ready to start (**enabled**state).

Syntax

```
If AX1.Drive.IsReadyForOperation then
...
end_if;
```

state administration

The property can be read in any state.

17 MCAxis.Drive.Motor

The **MCAxis.Drive.Motor** function allows you to read and adjust the properties of the MCX axis motor.

These are essentially the parameters available in the JetSym programming tool under "motion setup" on the "Commissioning" page, for Bucher-Automation Axes." When the controller is initialized, the parameters are set to the values saved during commissioning.

Using the Motion API commands, it is possible to read and adjust these parameters via the STX program as well.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

17.1 Syntax for MCAxis.Drive.Motor

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.Motor.<Function or Access to Value>(optional
<Parameter>);
```

17.2 Features of MCAxis.Drive.Motor

This function offers the following functions:

– MeasureCommutationOffset()

17.2.1 MeasureCommutationOffset()

This function measures the commutation offset for the MCX axis. The parameter passed defines the measurement method.

Measuring the commutation offset is necessary if, after power-up, the system does not know which value to select for the commutation offset. This is particularly true for axes equipped with incremental or sine-cosine encoders, as these encoders cannot measure the absolute angular position of the rotor.

NOTICE



Risk of unintentional startup

Calling the function causes the motor to generate torque and start moving. Take the following steps before using this feature:

- ► Make sure that the controller parameter settings—at a minimum, those for the current/torque controller—are correct.
- ► Make sure the motor shaft can turn freely.

transfer parameter

data type	Meaning
Enum DriveCommOffsetMeasurement-Modes (optional)	Specifies the measurement method for the commutation offset.
DriveCommOffsetMeasurement-Modes.StaticMode	The static method causes the motor to lock into two angular positions by applying current to it in a specific way. In this process, the motor shaft can move within the area of a pair of poles on the motor.
DriveCommOffsetMeasurement-Modes.DynamicMode	The dynamic method adjusts the rotating field while measuring the resulting motion. This method results in significantly reduced movement of the motor shaft, but requires more precise adjustment of the controllers.

Syntax

```
AX1.Drive.Motor.MeasureCommutationOffset(DriveCommOffsetMeasurement-
Modes.StaticMode);
```

state administration

ⓘ Info

Exclusive axis function

While the function is running, no other axis functions are available for this axis. The next STX command will not be executed until the measurement function has finished.

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	√	Inactive	Blocked
Approved	X		Approved
Working	X		Error: blocked
Interrupted	X		Approved
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3 Properties of MCAxis.Drive.Motor

This function offers the following properties:

– Engine Design
– Rated Current
– MaxCurrent
– Rated Speed
– Rated Torque
– Inductance

– Resistance
– VoltageConstant
– ConstantTorque
– PolePairs
– CommutationOffset
– ThermalDelayTime
– ShaftPosition
– Inertia
– TemperatureSensorType
– TemperatureSensorConnector
– Temperature
– I2tOperatingMode
– I2tValue

17.3.1 EngineDesign

Description

This property allows you to read the configured motor type.

With some servo amplifiers, in addition to a permanently excited synchronous motor (servo, another type can also be selected, such as a stepper motor.

data type	Meaning
DriveMotorDesignTypes (optional)	Engine design.
DriveMotorDesignTypes.Unknown	The drive cannot be operated. Readable only if initialization fails.
DriveMotorDesignTypes.PMSM_Servo (Default)	Permanently excited synchronous motor (servomotor). This is the most common type of motor used in servo applications.
DriveMotorDesignTypes.Stepper_2Ph_CL	Controlled two-phase stepper motor. This design requires a position sensor for control purposes.
DriveMotorDesignTypes.Stepper_2pH_OL	Uncontrolled two-phase stepper motor.
DriveMotorDesignTypes.Stepper_3pH_OL	Uncontrolled three-phase stepper motor.
DriveMotorDesignTypes.BLDC_3Ph	Electrically commutated three-phase DC motor.
DriveMotorDesignTypes.DC_HBridge	Permanently excited DC motor.
DriveMotorDesignTypes.ASM	Induction motor.
Drive Motor Design Types: Reluctance	Reluctance motor.

Syntax

```
MotorDesign := AX1.Drive.Amplifier.MotorDesign;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is running.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.2 RatedCurrent

Description

data type	Meaning
Double	Reads the rated current of the motor assigned to this axis in [A_{rms}].

Syntax

```
MotorRatedCurrent := AX1.Drive.Motor.RatedCurrent;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.3 MaxCurrent

Description

data type	Meaning
Double	Reads the maximum current of the motor assigned to this axis in [A_{rms}].

Syntax

```
MotorMaxCurrent := AX1.Drive.Motor.MaxCurrent;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.4 RatedSpeed

Description

data type	Meaning
Double	Reads the rated speed of the motor assigned to this axis in [rpm].

Syntax

```
MotorRatedSpeed := AX1.Drive.Motor.RatedSpeed;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved

Initial state	Allowed	Final state
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.5 RatedTorque

Description

data type	Meaning
Double	Reads the rated torque of the motor assigned to this axis in [Nm].

Syntax

```
MotorRatedTorque := AX1.Drive.Motor.RatedTorque;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.6 Inductance

Description

data type	Meaning
Double	Reads the inductance value between two phases of the motor associated with this axis in [mH].

Syntax

```
MotorInductance := AX1.Drive.Motor.Inductance;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.7 Resistance

Description

data type	Meaning
Double	Reads the electrical resistance between two phases of the motor associated with this axis in [Ω].

Syntax

```
MotorResistance := AX1.Drive.Motor.Resistance;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.8 VoltageConstant

Description

data type	Meaning
Double	Reads the value of the motor's voltage constant associated with this axis in [Vrms/ 1000 (rpm)].

Syntax

```
MotorVoltageConstant := AX1.Drive.Motor.VoltageConstant;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is running.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.9 ConstantTorque

Description

data type	Meaning
Double	Reads the torque constant of the motor assigned to this axis in [Nm/A].

Syntax

```
MotorTorqueConstant := AX1.Drive.Motor.TorqueConstant;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.10 PolePairs

Description

data type	Meaning
Double	Reads the number of pole pairs for the motor assigned to this axis.

Syntax

```
NumberOfMotorPolePairs := AX1.Drive.Motor.PolePairs;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.11 CommutationOffset

Description

The commutation offset tells the servo amplifier how the rotor's magnetic orientation relates to the encoder's zero point.

data type	Meaning
Double	Reads the commutation offset value of the motor assigned to this axis in electrical angle units [°].

Syntax

```
MotorCommutationOffset := AX1.Drive.Motor.CommutationOffset;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is running.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.12 ThermalDelayTime

Description

data type	Meaning
Double	Reads the value of the thermal time constant of the motor assigned to this axis in [s].

Syntax

```
MotorThermTimeConstant := AX1.Drive.Motor.ThermalDelayTime;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.13 ShaftPosition

Description

data type	Meaning
Double	Reads the position of the motor shaft associated with this axis in [0 … 360°].

Syntax

```
MotorRotorPosition := AX1.Drive.Motor.ShaftPosition;
```

state administration

The property can be read in any state.

17.3.14 Inertia

Description

data type	Meaning
Double	Reads the value of the motor's moment of inertia associated with this axis in [kg·cm²].

Syntax

```
MotorInertia := AX1.Drive.Motor.Inertia;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, since the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working

Initial state	Allowed	Final state
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.15 TemperatureSensorType

Description

This property allows you to read and write the type of the engine temperature sensor. The sensor type is not detected automatically; instead, it must be specified according to the motor's datasheet.

data type	Meaning
Type DriveTemperatureSensorTypes (optional)	Type of engine temperature sensor.
DriveTemperatureSensorTypes.NoSensor (Default)	No sensor is connected. The sensor is not being monitored, meaning that engine temperature monitoring is not active.
DriveTemperatureSensorTypes.NCC	Temperature switch; The engine temperature cannot be read; only the overtemperature monitoring system is active.
DriveTemperatureSensorTypes.PTC	Temperature-coefficient resistor with a nonlinear, positively sloping characteristic curve (PTC). The engine temperature cannot be read; only the overtemperature monitoring system is active.
DriveTemperatureSensorTypes.KTY83_110	KTY83-110 semiconductor sensor.
DriveTemperatureSensorTypes.KTY84_130	KTY84-130 semiconductor sensor.
DriveTemperatureSensorTypes.PT1000	Platinum measuring resistor.
DriveTemperatureSensorTypes.MotorSpecific	For specific motor-sensor combinations, such as the LinMot motor.

Syntax

```
MotorTempSensorType := AX1.Drive.Amplifier.TemperatureSensorType;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved

Initial state	Allowed	Final state
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.16 TemperatureSensorConnector

Description

This property allows you to read and write the port to which the engine temperature sensor is connected.

This property is available only on servo amplifiers in the JM-3000family, as these devices have three connectors for the motor temperature sensor: the motor connector and both encoder connectors on an axis.

data type	Meaning
Type: Drive Temperature Sensor Connectors (optional)	Selecting the connector for the engine temperature sensor.
Drive Temperature Sensor Connectors.Motor (Standard)	The engine temperature sensor is connected to the engine connector. **NOTICE! This data type is required for motors equipped with a Hiperface DSL position encoder.**
DriveTemperatureSensorConnectors.Enc1	The engine temperature sensor is connected to the position sensor connector labeled "Enc1."
DriveTemperatureSensorConnectors.Enc2	The engine temperature sensor is connected to the position sensor connector labeled "Enc2."

Syntax

```
MootrTempSensorConnection := AX1.Drive.Motor.TemperatureSensorConnector;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.17 Temperature

Description

data type	Meaning
Double	Reads the current temperature of the motor assigned to this axis in [°C].

Syntax

```
MotorTemperature := AX1.Drive.Motor.Temperature;
```

state administration

The property can be read in any state.

17.3.18 I2tOperatingMode

Description

data type	Meaning
DriveI2t Operating Modes (optional)	Status of the Motor I²t Monitoring
DriveI2tOperatingModes.Disable (Default)	I²t monitoring is disabled.
DriveI2tOperatingModes.Enable	I2T monitoring is enabled. For effective monitoring, the values for the thermal time constant [▶ 95] and the rated current [▶ 89] must be appropriate for the motor being used.

Syntax

```
MotorI2tActivation := AX1.Drive.Amplifier.I2tOperatingMode;
```

state administration

The property can be read in any state.

This property can also be accessed in write mode; however, this is not practical under normal circumstances, as the motor cannot be replaced while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

17.3.19 I2tValue

Description

This value is calculated only if the I²t calculation is enabled. For the value to be meaningful, the thermal time constant [▶ 95] values must match the rated current [▶ 89] of the motor being used.

data type	Meaning
Double	Reads the I²t integration value of the motor assigned to this axis in [%].

Syntax

```
MotorI2tValue := AX1.Drive.Amplifier.I2tValue;
```

state administration

The property can be read in any state.

18 MCAxis.Drive.QuickStopDeceleration

The **MCAxis.Drive.QuickStopDeceleration** function allows you to read or adjust the quick stop ramp of an MCX axis.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

18.1 Syntax for MCAxis.Drive.QuickStopDeceleration

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.QuickStopDeceleration;
```

18.2 Properties of MCAxis.Drive.QuickStopDeceleration

This function offers the following properties:

– QuickStop Deceleration

18.2.1 QuickStopDeceleration

Description

This property allows you to read or adjust the delay with which the axis is stopped when the `Power.Quick-stop` command is triggered. This ramp is also used in response to specific errors triggered in the servo amplifier.

The quick stop ramp is executed by the servo amplifier. The motion control function of the controller no longer affects the motion while the quick stop ramp is being executed.

The servo amplifier maintains speed control of the motor shaft using a linear deceleration ramp. This setting is intended to ensure that the mechanism moved by the MCX axis can be stopped just in time without causing damage.

data type	Meaning
Double	Reads or writes the value for the quick stop ramp delay in [rpm/s].

Syntax

```
EmergencyDeceleration := AX1.Drive.QuickStopDeceleration;
AX1.Drive.QuickStopDeceleration := 6000.0;
```

state administration

The property can be read and written in any state.

19 MCAxis.Drive.TorqueShutOff

The **MCAxis.Drive.TorqueShutOff** function allows you to stop an MCX axis very quickly when a torque threshold is reached and then control it to a specific position or holding torque. This function provides a complete sequence of actions based on torque and speed values.

ⓘ Info

Required adjustment of parameters while torque cut-off is active

The **TorqueShutOff** function intentionally affects the control of the servo amplifier in an unusual way. In addition, turning the screw brings the MCX axis to a near standstill without reaching the target position. The MCX setpoint generation is not affected by this. It continues to output position setpoints that correspond to the `MovePtp.Start`-command. As a result, both the position tracking error and the difference between the target and actual speeds may be greater than in normal positioning mode.

For this reason, it may be necessary to significantly increase the parameters for the slip error limit and speed deviation window, at least while the torque cut-off function is active.

ⓘ Info

Adjusting the pilot control while torque cut-off is active

Torque control can also cause the axis to behave erratically during operation. It is recommended that you disable this control, at least while the torque cut-off function is active. To do this, set the "Torque Pre-control Scaling" parameter to `0`.

ⓘ Info

Parameter behavior during direction reversal

If an MCX axis is operated with a reversed rotation direction, this has no effect on the sign of the configuration parameters for **the torque cut-off**function.

A typical application is capping bottles, especially plastic ones: When a certain torque is reached (the cap is unscrewed far enough), the motion must be stopped as quickly as possible, converting as much of the screwdriver head's kinetic energy into electrical energy as possible. This prevents the kinetic energy from being transferred to the screw connection and causing the lid to tighten further than intended. This ensures a more consistent tightening result.

This process is explained below using the example of screwing a cap onto a bottle. The process of tightening a screw connection is illustrated in the diagram below, with the actual*torque shut-off*occurring within the gray box.

There are two modes that determine what happens at the end of the process.

– In `HoldTorque` mode, a defined torque is applied at the end to hold the cap in place against the thread while it is stationary.
– In `HoldPosition` mode, the lid is controlled to the position reached at the torque threshold.

All values used in the commands refer to the motor shaft. A gearbox listed in the drive data has no effect on the values in the `TorqueTrigger`, `BrakeTorque`, `SpeedTrigger`, and `HoldingTorque` parameters. The units are in Nm for torque and in rpm for speed at the motor shaft.

The graph shows the qualitative variation of torque (orange) and rotational speed (blue) of the motor shaft over time:

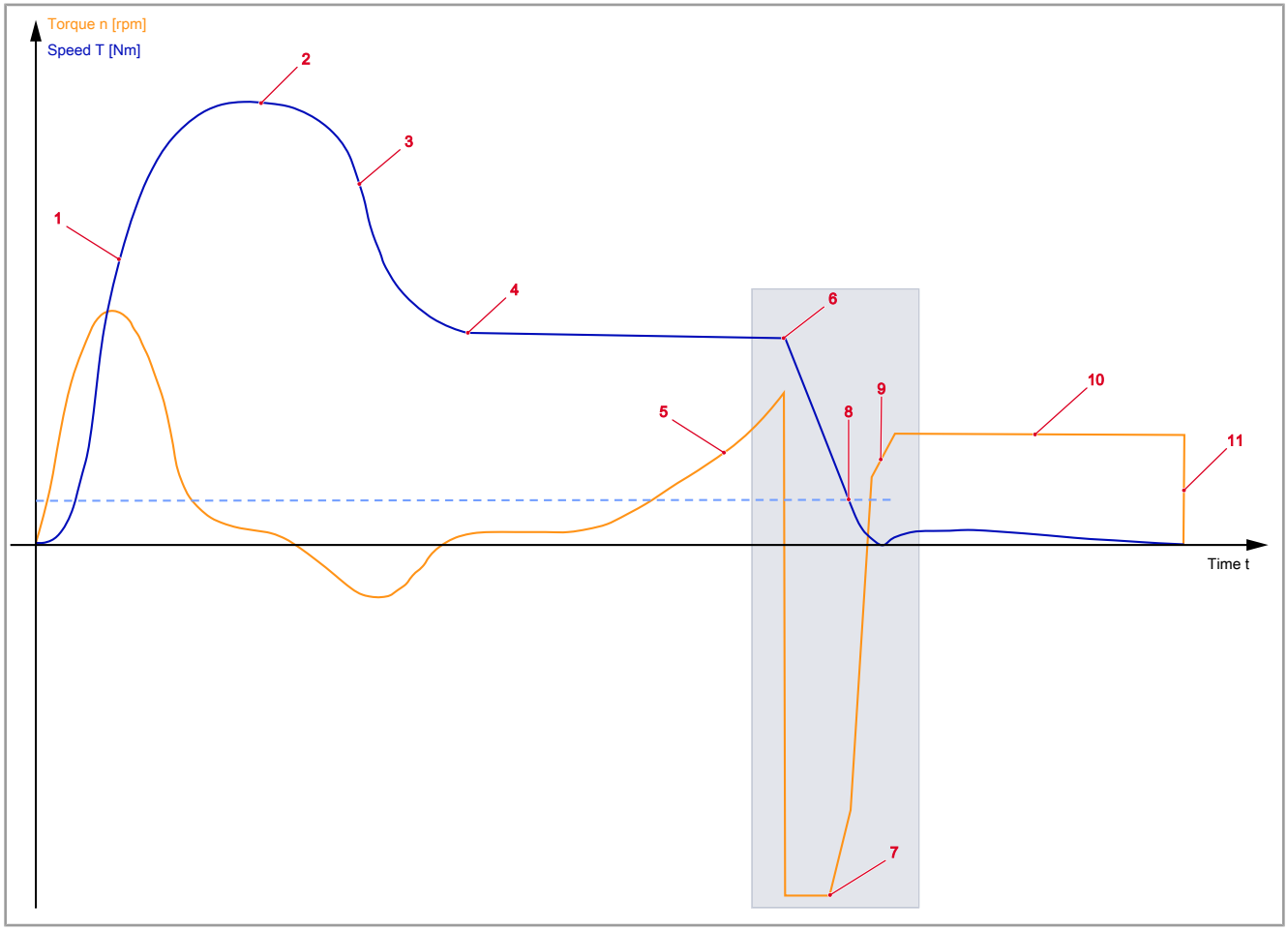


Fig. 1: MCX - Torque Shut-Off

No.	Description
1	Acceleration phase to high speed. Since this acceleration can generate a higher (dynamic) torque than the torque cut-off, the torque cut-off is not yet activated at this point.
2	Most of the thread is tightened at high speed.
3	Decelerate to low speed. This minimizes the energy stored in the inertia of the screwdriver head and the motor. This energy is directly dependent on the rotational speed.
4	Low speed reached: Torque cut-off is activated. This is where the actual process within the MCX function begins.
5	The lid is approaching the screw-on position: The torque is increasing.
6	Torque threshold (`TorqueTrigger`) reached: The `TorqueShutOff`function limits the target speed to 0. This causes the speed controller to reduce the speed as much as possible to the set torque limit. To assist this process, the I-component of the speed controller is described once using a “braking torque” (`BrakeTorque`) to initiate the braking process as quickly as possible. These two actions ensure that the screwdriver head is brought to a stop as quickly as possible, overcoming its inertia.
7	The difference between the target and actual speed decreases: The torque exceeds the limit.

No.	Description	
8	When the specified speed threshold (`SpeedTrigger`) is reached, the speed limit is lifted to prevent the motor from slowing down. Steps 9 and 10 depend on the mode:	
	Mode `HoldPosition`	**Mode** `HoldTorque`
9	A positive torque value (`holding torque`) is written to the I-component of the speed controller once. This affects the control system in such a way as to ensure a transition to position control that is as smooth as possible.	The torque limit is set to the value of `Holding-Torque`.
10	The target position is set to the position the shaft was in at the speed threshold: The lid is held in this position by the position control system.	The torque limit remains active until the torque cut-off function is deactivated. As a result, the lid may continue to move very slowly at a specified torque until it reaches its final position.
11	At the end of the tightening process, the bearing adjuster is set to the current position. This is necessary because the torque cut-off function has made changes to the controllers "below" the MCX setpoint generation, causing the actual position value to deviate from the setpoint rather than following it as usual. The `MCAxis.State.Transitions.UserPositionReset()` command allows you to set the target position to the actual position value, thereby resetting the position control to a defined initial position. Alternatively, you can lock the axis using the `MCAxis.Power.Disable()` command. When the release is re-engaged, the target position is also preset to match the actual position, so that no jerk occurs. In addition, in this case, all internal controllers start with a zero accumulated I-sum.	
	Finally, the torque cut-off function is deactivated again.	

19.1 Syntax for MCAxis.Drive.TorqueShutOff

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Drive.TorqueShutOff.<Function>(optional <Parameter>);
```

19.2 Features of MCAxis.Drive.TorqueShutOff

This function offers the following functions:

– Configure()
– ReadConfiguration()
– Activate()
– Deactivate()

19.2.1 Configure()

This function allows you to pass the information relevant to your application to the **Drive.TorqueShutOff** function.

transfer parameter

Parameters	data type	Meaning
Mode	Drive Torque Shut-Off Modes	Sets the control mode at the end of the cycle.
	DriveTorqueShutOff-Modes.HoldPosition	At the end of the cycle, returns to the position the axis was in when the speed threshold was reached.
	DriveTorqueShutOff-Modes.HoldTorque	Applies the value of `HoldingTorque` at the end of the process. Take the example of a screw-fastening process: The torque with which the axis "presses" against the thread.
TorqueTrig-ger	Double	Torque value at which the torque cut-off should be acti-vated, in [Nm at the motor shaft]. This value is an absolute value and is always positive.
BrakeTorque	Double	Torque used to preset the I component of the speed con-troller during rapid deceleration, in [Nm at the motor shaft]. This value must be negative if you want to provide very strong braking assistance.
SpeedTrig-ger	Double	Speed threshold below which the braking process is inter-rupted in order to continue the sequence depending on the mode, in [rpm at the motor shaft]. This value is an absolute value and is always positive.
Holding Torque	Double	– In `HoldPosition` mode: The torque used to preset the I-component of the speed controller in order to achieve the smoothest possible transition to position control after braking. – In `HoldTorque`mode: Torque that continues to act on the axis after braking. Using the example of a screw-tightening operation: the torque with which the axis “presses” further against the thread [in Nm at the motor shaft]. This value is an absolute value and is always positive.

Syntax

```
AX1.Drive.TorqueShutOff.Configure(DriveTorqueShutOffModes.HoldPosition, 1.5,
BrakeTorque, SpeedTrigger, HoldTorque);
```

state administration

The function can be called from any state.

19.2.2 ReadConfiguration()

This function reads the currently active configuration for the **Drive.TorqueShutOff** function.

transfer parameter

The function has no transfer parameters. The return value is returned in a structure of type `Drive-TorqueShutOffConfiguration`. It consists of the following elements:

Parameters	data type	Meaning
Mode	Drive Torque Shut-Off Modes	Sets the control mode at the end of the cycle.
	DriveTorqueShutOff-Modes.HoldPosition	At the end of the cycle, returns to the position the axis was in when the speed threshold was reached.
	DriveTorqueShutOff-Modes.HoldTorque	Applies the value of `HoldingTorque` at the end of the process. Take the example of a screw-fastening process: The torque with which the axis "presses" against the thread.
TorqueTrig-ger	Double	Torque value at which the torque cut-off should be acti-vated, in [Nm at the motor shaft]. This value is an absolute value and is always positive.
BrakeTorque	Double	Torque used to preset the I component of the speed con-troller during rapid deceleration, in [Nm at the motor shaft]. This value must be negative if you want to provide very strong braking assistance.
SpeedTrig-ger	Double	Speed threshold below which the braking process is inter-rupted in order to continue the sequence depending on the mode, in [rpm at the motor shaft]. This value is an absolute value and is always positive.
Holding-Torque	Double	– In `HoldPosition` mode: The torque used to preset the I-component of the speed controller in order to achieve the smoothest possible transition to position control after braking. – In `HoldTorque` mode: Applies the value of `HoldingTorque` at the end of the process. Take the example of a screw-fastening process: The torque with which the axis "presses" against the thread. This value is an absolute value and is always positive.

Syntax

```
CurrentConfiguration := Resol_3000.Drive.TorqueShutOff.ReadConfiguration();
```

state administration

The function can be read in any state.

19.2.3 Activate()

This function enables the **Drive.TorqueShutOff**function. The MCX axis then compares the torque at the motor shaft with the value configured in `TorqueTrigger` . As soon as the torque increases, the rapid braking function is activated.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.Drive.TorqueShutOff.Activate();
```

state administration

The function can be called from any state.

19.2.4 Deactivate()

This function disables the **Drive.TorqueShutOff** function. No more operations of the function will be exe-cuted. All torque and speed limits are reset to their original settings.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.Drive.TorqueShutOff.Deactivate();
```

state administration

The function can be called from any state.

19.3 Properties of MCAxis.Drive.TorqueShutOff

This function offers the following properties:

– State

19.3.1 State

Description

data type	Meaning
DriveTorqueShutOffStates (op-tional)	Reads the current status of the **Drive.TorqueShutOff** function.
DriveTorqueShutOffStates.Dis-abled	The feature is not enabled.
DriveTorqueShutOffStates.Wait-ForTorque	The feature has been enabled. The torque at the motor shaft is even lower than the value in `TorqueTrigger`.
DriveTorqueShutOffStates.Wait-ForSpeed	`The torque trigger` has been exceeded. Braking has begun. The RPM has not yet fallen below the value set in `SpeedTrigger` .
DriveTorqueShutOffStates.Fin-ished	The process is complete. Depending on the configured mode, control is based on either position or torque.

Syntax

```
When AX1.Drive.TorqueShutOff.State == DriveTorqueShutOffStates.Finished ,
Continue;
```

state administration

The property can be read in any state.

19.4 Example task MCAxis.Drive.TorqueShutOff

Here is an example task for a single axis that illustrates the process shown in the flowchart above:

```
Var
    bStartDA: Bool;
    BrakeTorque, SpeedTrigger, HoldTorque: Double;
```

```
    MaxThreadLength, TightenSpeed: Double;
End_Var;

Task tTorqueShutOff Autorun

    Resol_3000.Drive.Control.Speed.TorqueFeedForwardScale := 0.0;

If Resol_3000.State.IsDisabled then
        Resol_3000.Power.Enable();
        If Resol_3000.State.IsEnabled , continue;
    End_if;

Loop
    When bStartDA Continue;

    Resol_3000.Drive.TorqueShutOff.Configure(DriveTorqueShutOffModes.HoldPos
ition, 1.5, BrakeTorque, SpeedTrigger, HoldTorque);

    Resol_3000.MovePtp.Start(AxisPositioningModes.AbsNormal,
MaxThreadLength, TightenSpeed,10000,10000,0.1);
    When Resol_3000.Mechanism.Slope.IsAtConstantSpeed , continue;

    Resol_3000.Drive.TorqueShutOff.Activate();

    When Resol_3000.Drive.TorqueShutOff.State ==
DriveTorqueShutOffStates.Finished , Continue;

    Resol_3000.State.Transitions.UserPositionReset();
    Delay(4); //[optional] let controllers stabilize
    Resol_3000.Drive.TorqueShutOff.Deactivate();

    bStartDA := False;

End_Loop;
End_Task;
```

20 MCAxis.Info

The **MCAxis.Info** function allows you to identify the axis objects in MotionControl.

This information was entered by the user in the Hardware Manager and motion setup within the project.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

20.1 Syntax for MCAxis.Info

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Info.<Eigenschaft>;
```

20.2 Properties of the function

This function offers the following properties:

– Name
– ObjectID
– ClassID

20.2.1 Name

Description

data type	Meaning
String	Retrieves the name of the individual axis as defined in the hardware configuration. This is helpful if you want to use the name in the trace or on an external display.

Syntax

```
AxisName := AX1.Info.Name;
```

In this `case, AxisName` contains the string `"AX1"`.

state administration

The property can be read in any state.

20.2.2 ObjectID

Description

data type	Meaning
Integer	Retrieves the internal control number of the path group. This property is typically only required in applications that use the MC Manager's features. One possible use case is described in the corresponding section MCManager (MCMgr) [▶ 401] .

Syntax

```
Object ID := AX1.Info.ObjectID;
```

state administration

The property can be read in any state.

20.2.3 ClassID

Description

data type	Meaning
MCClassIDs	Queries the class of an MC object. An MC object can be an axis, a technology group, or a path group. This property is typically only required in applications that use the MC Manager's features. One possible use case is described in the corresponding section MCManager (MCMgr) [▶ 401] .

Syntax

```
ClassID := AX1.Info.ClassID;
```

state administration

The property can be read in any state.

20.3 Example task MCAxis.Info

Here is an example task for a single axis that includes various Motion API functions:

After the example task has been executed, the trace displays whether each axis is in the **"running"** state or not. The axes are listed by their axis names.

```
function TraceRunningState(mcAxis: pointer to MCAxis)
    if mcAxis.State.IsRunning then
        trace(mcAxis.Info.Name + 'is running$n');
    else
        trace(mcAxis.Info.Name + 'is not running$n');
    end_if;
end_function;

task Sample1 // example task
var
    nAxisIndex:    int;
end_var;
    for nAxisIndex := 0 to mcMgr.AxisCount - 1 , incrementing by 1 , do
        TraceRunningState(mcMgr.AxisList[nAxisIndex]);
    end_for;
end_task;
```

21 MCAxis.IsInTargetWindow

The **MCAxis.IsInTargetWindow** function allows you to check whether the actual position of the axis object is within the defined target window.

The target window is passed to the axis object either in MotionSetup or via the `MCAxis.MovePtp.Start()` function. As soon as the axis object reaches its target window, the `IsInTargetWindow` property is set `to true`. When a new positioning movement begins, the `IsInTargetWindow` property is `set to false`.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

21.1 Syntax for MCAxis.IsInTargetWindow

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.IsInTargetWindow
```

21.2 Properties of MCAxis.IsInTartegWindow

`IsInTargetWindow` is a property of the axis object.

21.2.1 IsInTargetWindow

Description

data type	Meaning
Bool	Checks whether the current position of the axis object is within the target window.

Syntax

```
If AX1.IsInTargetWindow , Continue;
```

state administration

The property can be queried in any state.

21.3 Example task MCAxis.IsInTargetWindow

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    run: bool; //Auxiliary flag for advancing in the program
End_Var;

Task Sample2 Autorun

loop
    when run Continue;

    //Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
```

```
    End_if;
    When AX1.State.IsEnabled , continue;

    AX1.MoveHome.SetReference(0.0); //Set the reference position to the
power-on position.
    When AX1.MoveHome.IsReferenceSet , continue;

    //Position the axis absolutely at 10,000 mm - Maximum speed 500 mm/s -
    //Max. Acceleration 1000 mm/s² - Max. deceleration 5000 mm/s²
    //Target window 0.1 mm

    AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,10000,500,1000,5000,0.1
);
    If AX1.IsInTargetWindow , Continue;

    //When the axis has reached its position, wait 1 second.
    Delay(t#1s);

    //Reposition the axis by the same distance; here, relative to the last
target position.
    AX1.MovePtp.StartNewTargetPosition(AxisPositioningModes.RelTarget,-10000
,);
    If AX1.IsInTargetWindow , Continue;

    //When the axis has arrived, wait 1 second.
    Delay(t#1s);

    //If the user has since cleared the auxiliary flag in the setup, lock
the axis and wait for it to be set again.
    //Otherwise, repeat the axis reversal.
    If Not Run Then

        AX1.Power.Disable();
        When AX1.State.IsDisabled , continue;
    End_if;

End_loop;

End_Task;
```

22 MCAxis.Mechanism.AxisMode

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

The **MCAxis.Mechanism.AxisMode** function allows you to retrieve the mode of an MCX axis as configured in the JetSym Hardware Manager.

The mode determines the function of the axis with respect to the various MCX functions, as well as how the axis can be used. Available axis modes:

– **InternalAxis**

  An internal axis is a real axis that is connected to the controller's system bus and, thanks to a motor and servo amplifier, is capable of moving a load.

  For an axis configured in this mode, all settings required for actual operation can be configured.

– **VirtualAxis**

  A virtual axis acts as an auxiliary axis that behaves like a real axis from the perspective of motion control, but cannot perform any actual movement. The actual position of this axis is always equal to the target position. The axis function is calculated solely within the controller.

  Due to the limited functionality, only mechanical parameters can be adjusted.

  The virtual axis is primarily used as an (ideal) reference axis for technology groups.

– **ShadowAxis**

  A drive shaft is used to cascade technology groups. The follow-up axis is "attached" to another (internal or virtual) axis; that is, it adopts all the parameters and behavior of its so-called reference axis. In particular, the position of the data axis serves as a reference axis for other technology groups. In this way, for example, the position of a secondary axis in one technology group can be used as the reference axis for another technology group (cascading).

  A driven shaft must always be used as the leading shaft in a technical assembly.

  Due to the limited functionality, only the units of the mechanical parameters can be changed.

– **ExternalAxis**

  An external axis is an MCX axis that simply reads the position data from a position sensor on the machine. The movement of this axis is not affected by the MCX controller. It can be used as the current position of this machine part in the STX program or as the reference axis position for a technology group.

  The encoder evaluated on the external axis must be connected to the second encoder interface of an internal axis. This internal axis must be configured so that the position of this encoder is read correctly and communicated to the MCX controller for external axes.

  In the axis parameters for the external axis, you must select which enabled second encoder is to be used for the external axis.

  Due to the limited functionality, only mechanical parameters can be adjusted.

– **SimulatedAxis**

  A simulated axis is an axis that behaves like an internal axis, but in which the servo amplifier and motor are simulated. With a simulation axis, there is no hardware access to the JetMove via the system bus.

– **SynchronousAxis**

  A synchronous axis is an axis that can be moved in sync with the path movement without being mechanically connected to it.

You can only change the configured mode in JetSym 's Hardware Manager. For the change to take effect, you must re-transfer the project configuration to the controller.

22.1 Syntax for MCAxis.Mechanism.AxisMode

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.AxisMode
```

22.2 Properties of MCAxis.Mechanism.AxisMode

`AxisMode` is a property of the `Mechanism` function.

22.2.1 AxisMode

For a detailed description of the axis modes, see *Application Note 057*: *MCX axes*.

Description

data type	Meaning
MCAxisModes (optional)	Configured mode of the axis
InternalAxis VirtualAxis SchadowAxis ExternalAxis SimulatedAxis	See the descriptions in the MCAxis.Mechanism.AxisMode [▶ 113]section.

Syntax

```
AxisMode := AX1.Mechanism.AxisMode;
```

state administration

The property can be read in any state.

23 MCAxis.Mechanism.AxisType

The **MCAxis.Mechanism.AxisType** function allows you to read and adjust the mechanical configuration of an MCX axis.

The control system must be aware of the mechanical configuration of an axis so that the units and kinematics can be calculated correctly in the MCX. This property determines whether the axis is infinite or finite, and whether a linear or non-linear mechanism is driven by the axis.

23.1 Syntax for MCAxis.Mechanism.AxisType

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.AxisType
```

23.2 Properties of MCAxis.Mechanism.AxisType

`AxisType` is a property of the `Mechanism` function.

23.2.1 AxisType

Use the Read or Set property to set the mechanical type of the axis.

For a detailed description of the mechanical axis types, see the section Description of Axis Types [▶ 116]."

data type	Meaning
AxisTypes (optional)	Sets the mechanical type of the axis.
Rotary	The axis configuration is rotational/normal.
RotatoryModulo	The axis configuration is rotational/modulo.
RotatoryModuloUniPos	The axis configuration is rotational/unidirectional/modulo-positive.
RotatoryModuloUniNeg	The axis configuration is rotational/unidirectional/modulo negative.
Linear	The axis configuration is linear/standard.
LinearModulo	The axis configuration is linear/modulo.
LinearModuloUniPos	The axis configuration is linear/unidirectional/positive modulo.
Linear Modulo One-Negative	The axis configuration is linear/unidirectional/modulo negative.
Crank	The axis is configured as a non-linear lever axis.
Piston	The axis is configured as a non-linear piston axis.

Syntax

```
AxisType := AX1.Mechanism.AxisType;
AX1.Mechanism.AxisType := AxisTypes.Crank;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked

Initial state	Allowed	Final state
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

23.3 Description of Axis Types

Mechanical axis types can be divided into four basic categories, each with its own subcategories:

– Rotational
– Linear
– Crank
– Piston

The choice between a rotary or linear axis is made in the JetSym MotionSetup (on the "Axis Parameters" page).

The Crank and Piston types are special mechanisms described at the end of this section.

23.3.1 Rotational axis

In a rotational axis, the connected load rotates around an axis. The motion parameters are therefore expressed in rotational units of measurement (e.g., degrees, °/s, etc.).

There may be a gear ratio that causes the number of motor revolutions to differ from the number of mechanical revolutions. The motion parameters (position, velocity, etc.) are always relative to the motion of the load.

Rotational/normal

This axis type is used when the load moves between two limits. This means that the load is moved in both directions. A maximum travel range can be specified for the load.

Rotational/modulo

This axis type has no limited travel range. The travel range defines only the so-called modulo travel range.

If the axis moves in the positive direction and reaches the positive travel range , the position is reset to the value of the negative travel range, and the axis can continue moving with new position values starting at the negative travel range.

If the axis moves in the negative direction and reaches the negative travel range, the position is reset to the value of the positive travel range, and the axis can continue moving with new position values starting at the positive travel range.

These axes are often used to always move in the same direction. Within the modulo travel range, the machine function is executed once (e.g., on a rotary indexing table).

Rotational/modulo/unidirectional

This axis type should be used when a load must be moved in only one direction of rotation (positive or negative).

Example:

With a unidirectional-positive axis, the load will move in the positive direction of rotation even if a target position smaller than the starting position is specified when the `MCAxis.MovePtP` command is called. In this case, the MCX moves the axis beyond the positive modulo travel range to the specified target position.

23.3.2 Linear axis

With a linear axis, the connected load moves in a straight direction. A rotary motor requires a mechanism that con-verts the rotating motion of the motor shaft (including any gear ratio) into linear motion of the load (e.g., a spindle). The motion parameters are therefore specified in linear units of measurement (e.g., mm, mm/s, etc.).

The motion parameters (position, velocity, etc.) always refer to the motion of the mechanical system.

Linear/normal

This axis type is used when the load moves between two limits. This means that the load is moved in both directions. A maximum travel range can be specified for the load.

Linear/modulo

This axis type has no limited travel range. The travel range defines only the so-called modulo travel range.

If the axis moves in the positive direction and reaches the positive travel range, the position is reset to the value of the negative travel range, and the axis can continue moving with new position values starting at the negative travel range.

If the axis moves in the negative direction and reaches the negative travel range, the position is reset to the value of the positive travel range, and the axis can continue moving with new position values starting at the positive travel range.

These axes are often used to always move in the same direction. Within the modulo travel range, the machine function is executed once (e.g., in the case of a conveyor belt).

Linear/modulo/unidirectional

This axis type should be used when a load must be moved in only one direction (positive or negative).

Example:

With a unidirectional-positive axis, the load will move in the positive direction even if a target position smaller than the starting position is specified when the `MCAxis.MovePtP` command is called. In this case, the MCX moves the axis beyond the positive modulo travel range to the specified target position.

23.3.3 Non-linear eccentric axis

This axis type is used when linear motion of the load is generated by a (rotating) eccentric.

The travel range is equal to twice the distance between the eccentric and the center of the rotating mechanism. Using the kinematic conversion in the MCX, you can program load-based values in linear units.

To use this axis type, the MCX must be configured as follows:

– Definition of the axis type
– Mechanical configuration using the command:

```
AX1.Mechanism.Kinematics.Crank.Configure(offset, InvertDirection, Radius);
```

The kinematic conversion in the MCX is based on the following formula:

linear position = ±(radius * cos(æ) - offset)

The fact is:

`Radius`: the distance between the eccentric and the center of the rotating mechanism.

`α is` the angle of rotation of the eccentric disc.

`Offset`: the shift in the linear position.

The `InvertDirection` parameter determines the sign before the parenthesis in the formula.

23.3.4 Non-linear piston axis

This axis type is used when linear motion of the load is generated by a (rotating) eccentric (crankshaft) connected to a piston rod.

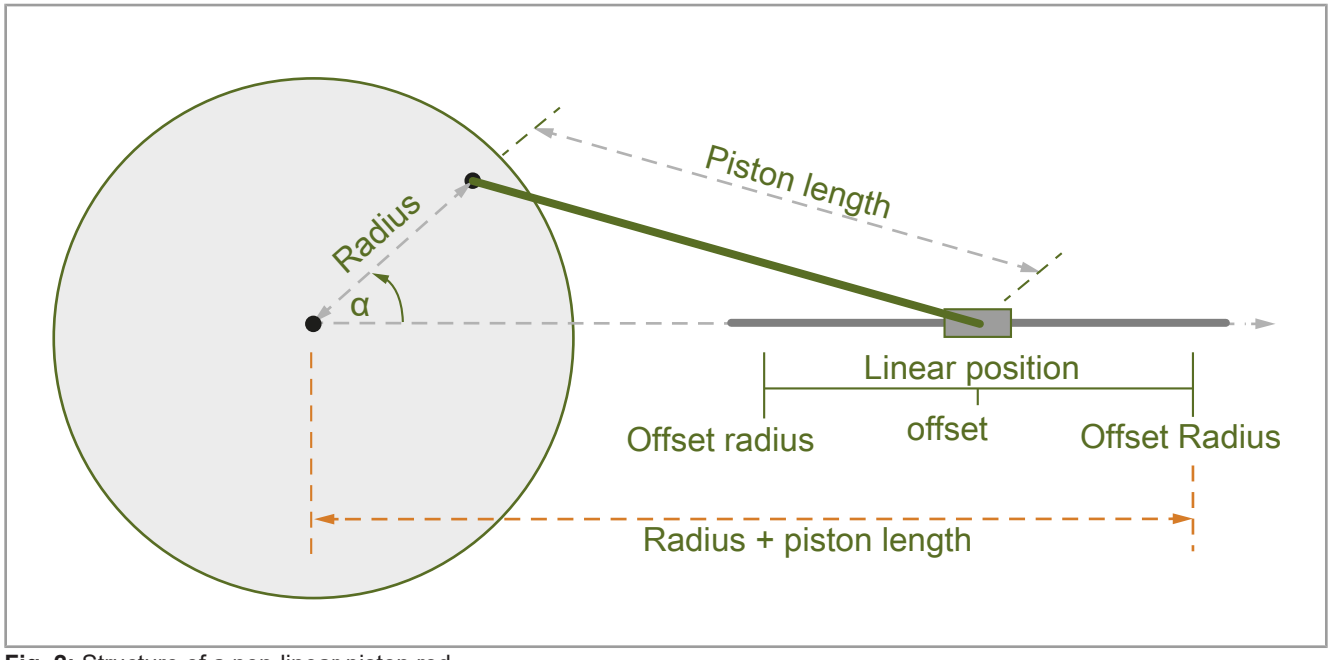


Fig. 2: Structure of a non-linear piston rod

The travel range is equal to twice the distance between the eccentric and the center of the rotating mechanism (radius). Using kinematics in the MCX, load-based programming can be performed in linear units of measurement.

To use this axis type, the MCX must be configured as follows:

– Definition of the axis type
– Mechanical configuration using the command:
  `AX1.Mechanism.Kinematics.Piston.Configure(offset, InvertDirection, Radius, RodLength);`
  The kinematic conversion in the MCX is based on the following formula:

$$\text{linear Position} = \pm\left(\text{Radius} * \cos(æ) + \sqrt{\text{RodLength}^2\text{-(Radius} * \sin(æ))^2} - \text{RodLength - Offset}\right)$$

The fact is:

Radius: the distance between the eccentric and the center of the rotating mechanism.

RodLength: the length of the piston rod.

α is the angle of rotation of the eccentric disc.

Offset: the shift in the linear position.

The `InvertDirection` parameter determines the sign before the parentheses in the formula.

23.4 Example task for MCAxis.Mechanism.AxisType

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample_AxisType Autorun
    …
If AX2.State.IsDisabled then;
    AX2.Mechanism.Kinematics.Piston.Configure(offset, InvertDir, Radius,
RodLength);
    AX2.Mechanism.AxisType := AxisTypes.Piston;
    End_If;
    …
If AX1.Mechanism.AxisType == AxisTypes.RotatoryModulo then
        AX1.MoveVelocity.Start(Directions.Positive, Speed, Accel, Decel);
    End_If;
    …
End_Task;
```

24 MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor

You can use the **MCAxis.Mechanism.GearRatioLoad** and **MCAxis.Mechanism.GearRatioMotor** functions to read and adjust the gear ratio of the gearbox being used.

ⓘ Info

If either of these properties is changed for an axis that has already been referenced, the reference is lost. The referencing run must be repeated.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

24.1 Syntax for MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.GearRatioLoad
```

and

```
<Axis name>.Mechanism.GearRatioMotor
```

24.2 Properties of MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor

`GearRatioLoad` and `GearRatioMotor` are properties of the `Mechanism` function.

24.2.1 GearRatioMotor

Description

data type	Meaning
Double	Specifies the motor-side shaft speed of the gearbox relative to the output speed. Only integer values are allowed.

Example

For a gear ratio of i = 3.75, set `GearRatioMotor` to 375 and `GearRatioLoad` to 100.

Syntax

```
AX1.Mechanism.GearRatioMotor := 375;
iMotor := AX1.Mechanism.GearRatioMotor;
```

state administration

The property can be read in any state.

The following table applies to the notation of gear ratios:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

24.2.2 Gear Ratio Load

Description

data type	Meaning
Double	Specifies the output shaft speed of the gearbox relative to the input speed. Only integer values are allowed.

Example

For a gear ratio of i = 3.75, set `GearRatioMotor` to 375 and `GearRatioLoad` to 100.

Syntax

```
AX1.Mechanism.GearRatioLoad := 375;
iLast := AX1.Mechanism.GearRatioLoad;
```

state administration

The property can be read in any state.

The following table applies to the notation of gear ratios:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

25 MCAxis.Mechanism.Kinematics

The **MCAxis.Mechanism.Kinematics** function allows you to read and adjust the mechanical configuration of a non-linearcrank [▶ 118]or non-linearpiston [▶ 118].

The control system must be aware of the mechanical configuration of an axis so that the units and kinematics can be calculated correctly in the MCX. The kinematic data must be provided before defining the MCX axis type. After defining the kinematics, you must change the axis type using the Motion API command MCAxis.Mechanism.AxisType.

A detailed description of the kinematic function of these two axis types under the command MCAxis.Mecha-nism.AxisType [▶ 115].

25.1 Syntax for MCAxis.Mechanism.Kinematics

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.Kinematics.<Function or Access to Value>(op-tional
<Parameter 1, Parameter 2, Parameter n>);
```

25.2 Functions of MCAxis.Mechanism.Kinematics

This function offers the following functions:

– Crank.Configure()
– Piston.Configure()

25.2.1 Crank.Configure(), Piston.Configure()

Description

This function allows you to pass all properties of an eccentric or piston axis in a single MCX command. Using these properties, the MCX can calculate the kinematic transformation. This allows you to program the desired linear motion of the load in linear units, even though the drive motor is rotary.

The properties of the function are read-only.

transfer parameter

Parameters / Properties	data type	Meaning
Offset	Double (optional)	Moves the linear position.
InvertDirection	Modes (optional)	Defines the counting direction of the linear position.
Radius	Double (optional)	Defines the distance between the axis center and the eccentric (or the piston rod mounting).
RodLength	Double (optional)	Only at Piston: Defines the length of the piston rod.

Syntax

```
AX1.Mechanism.Kinematics.Crank.Configure(15.5, Modes.On,25.0);
AX1.Mechanism.Kinematics.Piston.Configure(300.5, Modes.Off,200.0,115.3);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

25.3 Properties of MCAxis.Mechanism.Kinematics

This function offers the following properties:

– Crank.Radius
– Crank.Offset
– Crank.InvertDirection
– Piston.Radius
– Piston.Rod Length
– Piston.Offset
– Piston.InvertDirection

25.3.1 .Offset / .InvertDirection / .Radius / .RodLength

Description

Use the `offset`, `InvertDirection`, `Radius`, and `RodLength` properties to read the currently valid kinematic parameters of the eccentric or piston axis.

property	data type	Meaning
offset	Double (optional)	Moves the linear position.
InvertDirection	Modes (optional) Modes.On Modes.Off	Defines the counting direction of the linear position.
Radius	Double (optional)	Defines the distance between the shaft center and the eccentric (or the piston rod mounting).
Rod Length	Double (optional)	Only at Piston: Defines the length of the piston rod.

Syntax

```
EccentricOffset := AX1.Mechanism.Kinematics.Crank.Offset;
PistonAxisRadius := AX1.Mechanism.Kinematics.Piston.Radius;
PistonRodLength := AX1.Mechanism.Kinematics.Piston.RodLength;
ExctenerDirection := AX1.Mechanism.Kinematics.Crank.InvertDirection
```

state administration

The property can be read in any state.

To describe the properties, use the `Configure()` function.

25.4 Example task MCAxis.Mechanism.Kinematics

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample_AxisType Autorun

    …

    If AX2.State.IsDisabled then
        AX2.Mechanism.Kinematics.Piston.Configure(offset, InvertDir, Radius,
RodLength);
        AX2.Mechanism.AxisType := AxisTypes.Piston;
    End_If;

    …

    If AX1.Mechanism.AxisType == AxisTypes.RotatoryModulo then
        AX1.MoveVelocity.Start(Directions.Positive, Speed, Accel, Decel);
    End_If;

    …

End_Task;
```

26 MCAxis.Mechanism.Limitations

The **MCAxis.Mechanism.Limitations** function allows you to provide the MCX controller with information about the mechanical constraints of the MCX axis in the machine. These are used as limit values for motion programming.

If the values for the travel limits are exceeded when programming a movement, an error is generated and the movement is not started.

In the case of a modular axis, the position limits define the modular range. If the axis exceeds the `TravelLimitMax`value, it will continue moving to the `TravelLimitMin` value without stopping. The same applies: If the axis falls below the `TravelLimitMin` value, it will continue to the `TravelLimitMax` value without stopping.

When calculating the motion profile, the maximum values for velocity, acceleration, and in the case of jerk-limited motion types jerk are taken into account. If the motion command results in values for speed, acceleration, and jerk that exceed the limits, these values are automatically capped at the maximum limits without an error message.

26.1 Syntax for MCAxis.Mechanism.Limitations

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.Limitations.<Function or Access to Value>(op-tional
<Parameter 1, Parameter 2, Parameter n>);
```

26.2 Functions of MCAxis.Mechanism.Limitations

This function offers the following functions:

– SetTravelLimits()

26.2.1 SetTravelLimits(

Use this function to specify the travel range in load position units. This is determined by the smallest and largest position values within the travel range relative to the reference point.

For a modular axis, these two values define the module range. In the case of a modular axis, the position limits define the modular range. If the axis exceeds the `TravelLimitMax` value, it will move to the `TravelLimitMin` value without stopping. The same applies: If the axis falls below the `TravelLimitMin` value, the position is continued to the `TravelLimitMax` value without stopping.

ⓘ Info

Loss of reference due to execution of SetTravelLimits

After executing this function, the axis is no longer homed. A referencing run that was previously performed must be repeated. This does not apply to an axis with an absolute encoder that has been referenced using a reference offset (see SetAbsEncoder-Reference() [▶ 154]).

To read the values for the travel range/module range, use the `TravelLimitMin` and `TravelLimitMax`-properties.

transfer parameter

Parameters	data type	Meaning
TravelLimitMin	Double (optional)	Smallest position value within the travel range/module range; absolute value relative to the reference point.

Parameters	data type	Meaning
TravelLimitMax	Double (optional)	Largest position value within the travel range/module range; absolute value relative to the reference point.

Syntax

```
AX1.Mechanism.Limitations.SetTravelLimit(0.0, 360.0);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

26.3 Properties of MCAxis.Mechanism.Limitations

This function offers the following properties of the `Mechanism` function:

– TravelLimitMin
– TravelLimitMax
– MaxSpeed
– MaxAcceleration
– MaxJerk

26.3.1 TravelLimitMin

Description

data type	Meaning
Double (optional)	Reads or writes the smaller value for the travel or module range (see the SetTravelLimits() [▶ 125]function).

Syntax

```
TravelRangeLimitMin := AX1.Mechanism.Limitations.TravelLimitMin;
AX1.Mechanism.Limitations.TravelLimitMin := -100.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

26.3.2 TravelLimitMax

Description

Dentyp	Meaning
Double (optional)	Reads or writes the larger value for the travel or module range (see the SetTravel-Limits() [▶ 125]function).

Syntax

```
TravelLimitMax := AX1.Mechanism.Limitations.TravelLimitMax;
AX1.Mechanism.Limitations.TravelLimitMax := 1250.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

26.3.3 MaxSpeed

Description

Dentyp	Meaning
Double (optional)	Reads or writes the value for the maximum speed at which the load can or should be moved. This value may be determined, for example, by the machine's function or by the maximum specifications of a threaded spindle or other components involved in the drive system.

Syntax

```
MaximumSpeed := AX1.Mechanism.Limitations.MaxSpeed;
AX1.Mechanism.Limitations.MaxSpeed := 1.5; // m/s
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

26.3.4 MaxAcceleration

Description

Dentyp	Meaning
Double (optional)	Reads or writes the value for the maximum acceleration at which the load can or should be moved. This value may be determined, for example, by the machine's function or by the maximum specifications of a threaded spindle or other components involved in the drive system. The monitoring system applies this value to both the acceleration and deceleration phases of a movement.

Syntax

```
MaximumAcceleration := AX1.Mechanism.Limitations.MaxAcceleration;
AX1.Mechanism.Limitations.MaxAcceleration := 15000; // °/s²
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

26.3.5 MaxJerk

Description

Dentyp	Meaning
Double (optional)	Reads or writes the value for the maximum jerk at which the load can or should be moved. This value may be determined, for example, by the machine's function or by the maximum specifications of a threaded spindle or other components involved in the drive system.

Syntax

```
MaxJerk := AX1.Mechanism.Limitations.MaxJerk;
AX1.Mechanism.Limitations.MaxJerk := 20000.0; // m/s³
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

27 MCAxis.Mechanism.LinearRotationRatio

The **MCAxis.Mechanism.LinearRotationRatio** function allows you to read and adjust the gear ratio of mechanical components that convert rotational motor motion into linear motion (e.g., timing belts, lead screws, roller feeders). This function applies only to linearly configured axes.

The gear ratio is taken into account in such a way that these components take effect at the transmission output. In other words, if there is no gearbox, it must be configured with a 1:1 gear ratio.

27.1 Syntax for MCAxis.Mechanism.LinearRotationRatio

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.LinearRotationRatio
```

27.2 Properties of MCAxis.Mechanism.LinearRotationRatio

This function offers the following properties:

The **LinearRotationRatio** function is a property of the **Mechanism** function.

27.2.1 LinearRotationRatio

Description

Dentyp	Meaning
Double	Specifies the conversion of rotational motion into linear motion at the output side of the gearbox. In other words, what distance should the load travel when the gearbox output shaft completes one revolution? Example: For a ball screw with a pitch of 10 mm, the value to be specified is 10 mm/rev.

Syntax

```
AX1.Mechanism.LinearRotationRatio := 10.0;
LinearRotationRatio := AX1.Mechanism.LinearRotationRatio;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	–	Approved
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

28 MCAxis.Mechanism.Monitoring

The **MCAxis.Mechanism.Monitoring** function allows you to check the state of the limit switch inputs and the reference input of the MCX axis's servo amplifier.

The signals are displayed according to the set polarity. For example, an active hardware limit switch on an axis that is in motion will cause an error.

You can configure the signal polarity and the input mode in the properties of the MCAxis.Drive.DigitalInput [▶ 72] function.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

28.1 Syntax for MCAxis.Mechanism.Monitoring

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.Monitoring.<Access to Value>;
```

28.2 Properties of MCAxis.Mechanism.Monitoring

This function offers the following properties:

– IsReferenceSwitchActive
– IsHWLimitSwitchMinActive
– IsHWLimitSwitchMaxActive

28.2.1 IsReferenceSwitchEnabled

Description

data type	Meaning
Bool	Reads the status of the reference switch connected to the corresponding digital input on the servo amplifier. If the status is "active," the reference switch is detected during the referencing run for the MCX axis, and the appropriate action is taken if necessary.

Syntax

```
ReferenceSwitchStatus := AX1.Mechanism.Monitoring.IsReferenceSwitchActive;
```

state administration

The property can be read in any state.

It is not possible to configure this property, and therefore the signal function cannot be activated via software.

28.2.2 IsHWLimitSwitchMinActive

Description

This property allows you to read the status of the hardware limit switch connected to the corresponding digital input of the servo amplifier in the negative direction of travel.

If the status is *active*, an error is generated and the axis is stopped and locked. The axis can only be positioned in the opposite direction.

data type	Meaning
Bool	Treats the hardware limit switch in the negative travel direction as *inactive* or *active* .

Syntax

```
NegEndSwitchStatus := AX1.Mechanism.Monitoring.IsHWLimitSwitchMinActive;
```

state administration

The property can be read in any state.

28.2.3 IsHWLimitSwitchMaxActive

Description

This property allows you to read the status of the hardware limit switch connected to the corresponding digital input of the servo amplifier in the positive direction of travel.

If the status is *active*, an error is generated and the axis is stopped and locked. The axis can only be positioned in the opposite direction.

data type	Meaning
Bool	Treats the hardware limit switch in the positive travel direction as *inactive* or *active* .

Syntax

```
PosEndSwitchStatus := AX1.Mechanism.Monitoring.IsHWLimitSwitchMaxActive;
```

state administration

The property can be read in any state.

29 MCAxis.Mechanism.Slope

The **MCAxis.Mechanism.Slope** function allows you to read or adjust the calculated properties of the current or next motion (Config.RampBreakMode [▶ 140]).

A positioning movement consists of the following phases:

1. acceleration ramp
2. Driving at a constant speed
   This phase occurs only if the speed can be reached using the specified MovePtp parameters. Otherwise, the acceleration ramp transitions directly into the deceleration ramp (triangular mode).
3. deceleration ramp

In addition, no movement can occur if the axis is at its destination and no further positioning has been initiated.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

29.1 Syntax for MCAxis.Mechanism.Slope

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.Slope.<Access to Value>
```

29.2 Properties of MCAxis.Mechanism.Slope

This function offers the following properties:

– SlopeState
– Is Accelerating
– IsAtConstantSpeed
– IsDecelerating
– IsStopped
– SlopeType
– ActualMaxSpeed
– ActualMaximumAcceleration
– ActualMaximumDeceleration
– SpeedSetpoint
– AccelerationSetpoint
– Speed
– Acceleration
– Deceleration
– Config.RampBreakMode

29.2.1 SlopeState

Description

data type	Meaning
AxisSlopeStates	Reads the current phase of motion for the calculated setpoint of the MCX axis. This allows you, for example, to activate an action in the machine only after the MCX axis has exited the acceleration phase.
AxisSlopeStates.Accelerating	The axis is on the acceleration ramp.
AxisSlopeStates.AtConstantSpeed	The axis moves at a constant speed.
AxisSlopeStates.Decelerating	The axis is in the deceleration ramp.
AxisSlopeStates.Stopped	The axis is stationary.

Syntax

```
SlopeStatus := AX1.Mechanism.Slope.SlopeState;
```

state administration

The property can be read in any state.

29.2.2 IsAccelerating

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the MCX axis is within the acceleration ramp.

Syntax

```
If AX1.Mechanism.Slope.IsAccelerating then …
```

state administration

The property can be queried in any state.

29.2.3 IsAtConstantSpeed

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the MCX axis is in the constant-speed phase.

Syntax

```
If AX1.Mechanism.Slope.IsAtConstantSpeed then …
```

state administration

The property can be queried in any state.

29.2.4 IsDecelerating

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the MCX axis is within the deceleration ramp.

Syntax

```
If AX1.Mechanism.Slope.IsDecelerating , then …
```

state administration

The property can be queried in any state.

29.2.5 IsStopped

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the MCX axis is in the stop phase.

Syntax

```
If AX1.Mechanism.Slope.IsStopped then …
```

state administration

The property can be queried in any state.

29.2.6 SlopeType

Description

The `SlopeType` property allows you to read or adjust the ramp during the acceleration and deceleration phases. **NOTICE! This setting applies to all positioning movements of the MCX axis until the value is changed.**

data type	Meaning
Axis Slope Types	Shape of the acceleration and deceleration ramp.
AxisSlopeTypes.Sine	This setting calculates a motion that produces relatively little jerk, meaning the motion is smoother. Since the acceleration value passed to the motion command is only reached at the midpoint of the ramp, the ramp duration and length are longer than with the `linear`setting. The motion follows a sinusoidal path during the acceleration phase and a sin²-shaped path during the deceleration phase. The movement is limited to a certain extent.
AxisSlopeTypes.Linear	This setting allows for motion with the lowest possible maximum acceleration while maintaining the same ramp time (and length) as the `"sine"`setting. However, the jolt during the transition from a standstill to the acceleration phase and from the acceleration phase to the maximum speed phase is infinite. The motion is constant during the acceleration phase and sinusoidal during the deceleration phase. The movement occurs without any jerk limitation.

Syntax

```
RampType := AX1.Mechanism.Slope.SlopeType;
AX1.Mechanism.Slope.SlopeType := AxisSlopeTypes.Linear;
```

state administration

The property can be read and written in any state.

29.2.7 ActualMaxSpeed

Description

data type	Meaning
Double	Reads the maximum speed value currently programmed in the MCX axis's setpoint generator for the current or upcoming movement. This value can be changed by specifying a new speed value in the commands `MCAxis.MoveVelocity.Start()`, `MCAxis.MoveVelocity.NewSpeed()`, `MCAxis.MovePtp.Start()`, and `MCAxis.MovePtp.NewSpeed()` . If the parameter is not specified in these commands, the value returned by `ActualMaxSpeed` is used in the motion.

Syntax

```
MaxSpeed := AX1.Mechanism.Slope.ActualMaxSpeed;
```

state administration

The property can be read in any state.

29.2.8 ActualMaximumAcceleration

Description

data type	Meaning
Double	Reads the value of the maximum acceleration that was last passed to the MCX axis's setpoint generator. This value can be changed by specifying a new velocity value in the commands `MCAxis.MoveVelocity.Start()`, `MCAxis.MoveVelocity.NewAcceleration()`, `MCAxis.MovePtp.Start()`, and `MCAxis.MovePtp.NewAcceleration()` . If the parameter is not specified in these commands, the value returned by `ActualMaxAcceleration` is used in the motion.

Syntax

```
MaxAcceleration := AX1.Mechanism.Slope.ActualMaxAcceleration;
```

state administration

The property can be read in any state.

29.2.9 ActualMaximumDeceleration

Description

data type	Meaning
Double	Reads the value of the maximum acceleration that was last passed to the MCX axis's setpoint generator. This value can be changed by specifying a new velocity value in the commands `MCAxis.MoveVelocity.Start()`, `MCAxis.MoveVelocity.NewDeceleration()`, `MCAxis.MovePtp.Start()`, and `MCAxis.MovePtp.NewDeceleration()`. If the parameter is not specified in these commands, the value returned by `ActualMaxDeceleration` is used in the motion.

Syntax

```
MaxDelay := AX1.Mechanism.Slope.ActualMaxDeceleration;
```

state administration

The property can be read in any state.

29.2.10 AccelerationSetpoint

Description

data type	Meaning
Double	Reads the value of the setpoint acceleration currently active in the MCX axis's setpoint generator. The sign of the value indicates acceleration or deceleration.

Syntax

```
ActSetpoint := AX1.Mechanism.Slope.AccelerationSetpoint;
```

state administration

The property can be read in any state.

29.2.11 SpeedSetpoint

Description

data type	Meaning
Double	Reads the value of the setpoint speed currently active in the MCX axis's setpoint generator. The sign of the value depends on the direction of travel.

Syntax

```
ActSpeedSetpoint := AX1.Mechanism.Slope.SpeedSetpoint;
```

state administration

The property can be read in any state.

29.2.12 Speed

Description

data type	Meaning
Double	Reads or writes the value for the maximum speed used to generate the position setpoint for a positioning profile. This value is typically passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` , or in `MovePtp.NewSpeed()` and `MoveVelocity.NewSpeed()` , where a change is applied to a movement that is already in progress. It's different here: Regardless of the state in which the change to the `speed` value is passed, the new value does not take effect until the next movement.

Syntax

```
ProfileSpeed := AX1.Mechanism.Slope.Speed;
AX1.Mechanism.Slope.Speed := 3000.0;
```

state administration

The property can be read and written in any state.

29.2.13 Acceleration

Description

data type	Meaning
Double	Reads or writes the value for the maximum acceleration used to generate the position setpoint for a positioning profile. This value is typically passed as a parameter in the `MovePtp.Start()` and `Move.Velocity.Start()` functions, or in the `MovePtp.NewAcceleration()` and `MoveVelocity.NewAcceleration()` functions. Regardless of the state in which the change to the `acceleration` value is passed, the new value does not take effect until the next movement.

Syntax

```
ProfileAcceleration := AX1.Mechanism.Slope.Acceleration;
AX1.Mechanism.Slope.Acceleration := 5500.0;
```

state administration

The property can be read and written in any state.

29.2.14 Deceleration

Description

data type	Meaning
Double	Reads or writes the value for the maximum delay used to generate the position setpoint for a positioning profile. This value is typically passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` , or in `MovePtp.NewDeceleration()` and `MoveVelocity.NewDeceleration()` . Regardless of the state in which the change to the `deceleration` value is passed, the new value will not take effect until the next movement.

Syntax

```
ProfileDeceleration := AX1.Mechanism.Slope.Deceleration;
AX1.Mechanism.Slope.Acceleration := 12000.0;
```

state administration

The property can be read and written in any state.

29.2.15 Config.RampBreakMode

Description

Deceleration slope using MovePtp.NewDeceleration()

If the deceleration is not specified using the `AX1.MovePtp.Start()` command but instead using `AX1.MovePtp.NewDeceleration()`, positioning is always performed `immediately` according to the ramp configuration.

data type	Meaning
MCAxisRampBreakModes	Reads or writes the setpoint generation behavior when a new `MovePtp.Start()` command is issued while the motion is in the deceleration ramp. The value after power-on is `None`. This is a static configuration that remains active until it is changed.
MCAxisRampBreakModes.Immediate	1. The stopping distance is sufficient: The deceleration ramp is terminated using the newly specified maximum deceleration, and the second target position is approached directly. 2. The stopping distance is insufficient: The first target position is approached. Positioning to the second target position follows immediately.
MCAxisRampBreakModes.None	The first target position is approached. Positioning to the second target position follows immediately.

Syntax

```
RampConfiguration := AX1.Mechanism.Slope.Config.RampBreakMode;
AX1.Mechanism.Slope.Config.RampBreakMode := MCAxisRampBreakModes.Immediate;
```

state administration

The property can be read and written in any state.

30 MCAxis.Mechanism.Units

The **MCAxis.Mechanism.Units** function allows you to read or adjust the units used for position, velocity, acceleration, and jerk values.

These units are typically defined during axis definition and can be queried using these commands in the STX program.

NOTICE



Changes in position due to a change of unit during machine operation

It is possible to change the units while the machine is running. However, since the positioning command contains only the values without specifying a unit, this changes the meaning of the values.

Example: Converting from millimeters to inches results in a different position in the machine for a value of `25.0` in the positioning command.

- ► Avoid changing units during operation.
- ► Be careful when converting units.

30.1 Syntax for MCAxis.Mechanism.Units

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Mechanism.Units.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

30.2 Functions of MCAxis.Mechanism.Units

This function offers the following functions:

– SetLinearUnits()

– SetRotatoryUnits()

30.2.1 SetLinearUnits()

This function converts all units for position, velocity, acceleration, and jerk into a single unit for linear motion. You can use the parameters to assign different meanings to each unit.

transfer parameter

Parameters	data type	Meaning
PositionUnit	**MCLinearPositionUnits** (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for linear position values. Possible units: – millimeter – Customs – 1/10 millimeter – 1/100 millimeter

Parameters	data type	Meaning
SpeedUnit	**MCLinearSpeedUnits** (optional) – MCLinearSpeedUnits.MM_PER_SEC – MCLinearSpeedUnits.MM_PER_MIN – MCLinearSpeedUnits.M_PER_SEC – MCLinearSpeedUnits.M_PER_MIN – MCLinearSpeedUnits.INCH_PER_SEC – MCLinearSpeedUnits.INCH_PER_MIN	Unit for linear velocity values. Possible units: – millimeters per second – millimeters per minute – meters per second – meters per minute – Inches per second – Inches per minute
AccelerationUnit	**MCLinearAccelerationUnits** (optional) – MCLinearAccelerationUnits.MM_PER_S2 – MCLinearAccelerationUnits.M_PER_S2 – MCLinearAcceleratio-nUnits.INCH_PER_S2	Unit for linear acceleration values. Possible units: – millimeters per second² – meters per second² – inches per second²
JerkUnit	**MCLinearJerkUnits** (optional) – MCLinearJerkUnits.MM_PER_S3 – MCLinearJerkUnits.M_PER_S3 – MCLinearJerkUnits.INCH_PER_S3	Unit for linear jerk values. Possible units: – millimeters per second³ – meters per second³ – Inches per second³

Syntax

```
AX1.Mechanism.Units.SetLinearUnits(
    MCLinearPositionUnits.MM,
    MCLinearSpeedUnits.MM_PER_SEC,
    MCLinearAccelerationUnits.MM_PER_S2,
    MCLinearJerkUnits.MM_PER_S3);
```

state administration

The function can be read in any state.

The function can be written in any state. However, it is recommended that you only modify units while they are in the **Locked** state.

30.2.2 SetRotaryUnits()

This function converts all units for positions, velocities, accelerations, and jerks into a single function for rotational motion. You can use the parameters to assign different meanings to each unit.

transfer parameter

Parameters	data type	Meaning
PositionUnit	**MCRotatoryPositionUnits** (optional) – MCRotatoryPositionUnits.DEG – MCRotatoryPositionUnits.RAD – MCRotatoryPositionUnits.REV	Unit for the rotational position values. Possible units: – degree – Radiant (2-piece) – revolutions

Parameters	data type	Meaning
SpeedUnit	**MCRotarySpeedUnits** (optional) – MCRotarySpeedUnits.deg_per_sec – MCRotatorySpeedUnits.RAD_PER_SEC – MCRotarySpeedUnits.RPM	Unit for rotational speed values. Possible units: – degrees per second – Radiant (2 Pi) per second – revolutions per minute
AccelerationUnit	**MCRotatoryAccelerationUnits** (optional) – MCRotatoryAccelerationUnits.DEG_PER_S2 – MCRotatoryAccelerationUnits.RAD_PER_S2	Unit for rotational acceleration values. Possible units: – degrees per second² – Radiant (2 Pi) per square second
JerkUnit	**MCRotatoryJerkUnits** (optional) – MCRotatoryJerkUnits.DEG_PER_S3 – MCRotatoryJerkUnits.RAD_PER_S3	Unit for rotational jerk values. Possible units: – Degrees per second³ – Radiant (2 Pi) per second³

Syntax

```
AX1.Mechanism.Units.SetRotatoryUnits(
    MCRotatoryPositionUnits.DEG,
    MCRotarySpeedUnits.RPM,
    MCRotatoryAccelerationUnits.DEG_PER_S2,
    MCRotatoryJerkUnits.DEG_PER_S3);
```

state administration

The function can be read in any state.

The function can be written in any state. However, it is recommended that you only modify units while they are in the **Locked** state.

30.3 Properties of MCAxis.Mechanism.Units

This function offers the following properties:

– LinearPositionUnit
– LinearSpeedUnit
– LinearAccelerationUnit
– LinearJerkUnit
– RotaryPositionUnit
– RotarySpeedUnit
– RotaryAcceleration Unit
– RotatoryJerkUnit

30.3.1 LinearPositionUnit

Description

Reads or writes the unit in which the position values for a linearly defined axis are processed and displayed.

data type	Meaning
MCLinearPositionUnits (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for linear position values. Possible units: – millimeter – Customs – 1/10 millimeter – 1/100 millimeter

Syntax

```
PositionUnits := AX1.Mechanism.Units.LinearPositionUnit;
AX1.Mechanism.Units.LinearPositionUnit := MCLinearPositionUnits.MM_10TH;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.2 LinearSpeedUnit

Description

Reads or writes the unit in which speed values for a linearly defined axis are processed and displayed.

data type	Meaning
MCLinearSpeedUnits (optional) – MCLinearSpeedUnits.MM_PER_SEC – MCLinearSpeedUnits.MM_PER_MIN – MCLinearSpeedUnits.M_PER_SEC – MCLinearSpeedUnits.M_PER_MIN – MCLinearSpeedUnits.INCH_PER_SEC – MCLinearSpeedUnits.INCH_PER_MIN	Unit for linear velocity values. Possible units: – millimeters per second – millimeters per minute – meters per second – meters per minute – Inches per second – Inches per minute

Syntax

```
WeightUnits := AX1.Mechanism.Units.LinearSpeedUnit;
AX1.Mechanism.Units.LinearSpeedUnit := MCLinearSpeedUnits.M_SEC;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.3 LinearAccelerationUnit

Description

Reads or writes the unit in which acceleration values for a linearly defined axis are processed and displayed.

data type	Meaning
MCLinearAccelerationUnits (optional) – MCLinearAccelerationUnits.MM_PER_S2 – MCLinearAccelerationUnits.M_PER_S2 – MCLinearAccelerationUnits.INCH_PER_S2	Unit for linear velocity values. Possible units: – millimeters per second² – meters per second squared – inches per second²

Syntax

```
AccelerationUnits := AX1.Mechanism.Units.LinearAccelerationUnit;
AX1.Mechanism.Units.LinearAccelerationUnit :=
MCLinearAccelerationUnits.M_PER_S2;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.4 LinearJerkUnit

Description

Reads or writes the unit in which the jerk values for a linearly defined axis are processed and displayed.

data type	Meaning
MCLinearJerkUnits (optional) – MCLinearJerkUnits.MM_PER_S3 – MCLinearJerkUnits.M_PER_S3 – MCLinearJerkUnits.INCH_PER_S3	Unit for linear velocity values. Possible units: – millimeters per second³ – meters per second³ – Inches per second³

Syntax

```
JerkUnits := AX1.Mechanism.Units.LinearJerkUnit;
AX1.Mechanism.Units.LinearJerkUnit := MCLinearJerkUnits.MM_PER_S3;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.5 RotaryPositionUnit

Description

Reads or writes the unit in which the position values for a rotary axis are processed and displayed.

data type	Meaning
MCRotatoryPositionUnits (optional) – MCRotatoryPositionUnits.MM – MCRotatoryPositionUnits.INCH – MCRotatoryPositionUnits.MM_10TH – MCRotatoryPositionUnits.MM_100TH	Unit for the rotational position values. Possible units: – millimeter – Customs – 1/10 millimeter – 1/100 millimeter

Syntax

```
PositionUnits := AX1.Mechanism.Units.RotatoryPositionUnit;
AX1.Mechanism.Units.RotatoryPositionUnit := MCRotatoryPositionUnits.RAD;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.6 RotarySpeedUnit

Description

Reads or writes the unit in which the speed values for a rotationally defined axis are processed and displayed.

data type	Meaning
MCRotarySpeedUnits (optional) – MCRotarySpeedUnits.deg_per_sec – MCRotatorySpeedUnits.RAD_PER_SEC – MCLinearSpeedUnits.RPM	Unit for linear velocity values. Possible units: – degrees per second – Radiant (2 Pi) per second – revolutions per minute

Syntax

```
RotationalSpeedUnits := AX1.Mechanism.Units.RotatorySpeedUnit;
AX1.Mechanism.Units.RotatorySpeedUnit := MCRotatorySpeedUnits.REV_PER_MIN;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.7 RotaryAccelerationUnit

Description

Reads or writes the unit in which the angular acceleration values for a rotationally defined axis are processed and displayed.

data type	Meaning
MCRotatoryAccelerationUnits (optional) – MCRotatoryAccelerationUnits.DEG_PER_S2 – MCRotatoryAccelerationUnits.RAD_PER_S2	Unit for rotational acceleration values. Possible units: – degrees per second² – Radiant (2 Pi) per square second

Syntax

```
AccelerationUnits := AX1.Mechanism.Units.LinearAccelerationUnit;
AX1.Mechanism.Units.LinearAccelerationUnit :=
MCLinearAccelerationUnits.M_PER_S2;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

30.3.8 RotatoryJerkUnit

Description

Reads or writes the unit in which the jerk values for a rotationally defined axis are processed and displayed.

data type	Meaning
MCRotatoryJerkUnits (optional) – MCRotatoryAcceleratio-nUnits.DEG_PER_S3 – MCRotatoryAcceleratio-nUnits.RAD_PER_S3	Unit for rotational acceleration values. Possible units: – Degrees per second³ – Radiant (2 Pi) per second³

Syntax

```
JerkUnits := AX1.Mechanism.Units.RotatoryJerkUnit;
AX1.Mechanism.Units.RotatoryJerkUnit := MCRotatoryJerkUnits.MM_PER_S3;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

31 MCAxis.MoveHalt

The **MCAxis.MoveHalt** function allows you to stop an MCX axis that is in motion using a ramp. Unlike the `Stop()` function, the input buffers are not cleared. In buffered mode, this function is written to the buffer and processed there.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

31.1 Syntax for MCAxis.MoveHalt

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.MoveHalt.Start.(optional <Parameter 1, Parameter 2>);
```

31.2 Functions of MCAxis.MoveHalt

This function offers the following functions:

– Start()

31.2.1 Start()

This function executes the stop command immediately. The parameters passed define how the axis should stop.

Please follow these rules:

– The `Deceleration` parameter is entered in the user units that are predefined for the axis.

– Adjusting the maximum delay using this parameter changes the value in the axis object. For example, if no value is passed for the `Deceleration` parameter in the next `MovePtp.Start`command, the value specified here will be used.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

transfer parameter

Parameters	data type	Meaning
MCAxisHaltMode	Enum MCAxisHaltModes (optional)	Behavior of the axis after it has been slowed down us-ing a deceleration ramp:
	MCAxisHaltModes.Normal	Once the deceleration ramp is complete, the axis comes to a stop under position control.
	MCAxisHaltModes.AtActual-Position	After the stop ramp is complete, the axis returns to the position it was in when the `MoveHalt.Start()` function was called. The reverse movement is performed using the parameters set for this axis object.

Parameters	data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the deceleration ramp. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Syntax

```
AX1.MoveHalt.Start(MCAxisHaltModes.Normal,50000);
AX1.MoveHalt.Start(MCAxisHaltModes.AtActualPosition,50000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	–		Blocked
Approved	–		Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis has been stopped (MCAxis.State [▶ 200]).

```
When Ax1.State.IsEnabled Continue; //The axis is in position control mode
and is stationary.
When Ax1.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **ramp status** indicates which phase of motion the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 134]).

```
When Ax1.Mechanism.Slope.IsAccelerating Continue;
When Ax1.Mechanism.Slope.IsAtConstantSpeed Continue;
When Ax1.Mechanism.Slope.IsDecelerating , continue;
When Ax1.Mechanism.Slope.IsStopped , continue;
```

31.3 Example Task: MCAxis.MoveHalt

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    run: bool; //Auxiliary flag for advancing in the program
End_Var;

Task Sample2 Autorun

loop
    when run Continue;

    //Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled Continue;

    AX1.MoveHome.SetReference(0.0); //Set reference position to start
position.
    When AX1.MoveHome.IsReferenceSet Continue;

    //Start the axis in the positive direction in continuous operation.
    AX1.MoveVelocity.Start(Directions.Positive, Speed1,5000,);
    //If the user has since cleared the auxiliary flag in the setup, stop
the axis with a ramp of 500 mm/s² and lock it.
    //This event could be, for example, a push button.
    If not run , continue;
    AX1.MoveHalt.Start(MCAxisHaltModes.Normal,500);
    When AX1.State.IsDisabled Continue;
    AX1.Power.Disable();
    When AX1.State.IsDisabled , continue;

End_loop;

End_Task;
```

32 MCAxis.MoveHome

The **MCAxis.MoveHome** function allows you to start and/or configure referencing for an MCX axis.

The JetSym online help provides a detailed description of the different types of referencing runs, as well as their functions and procedures, on the *"MotionHome Flowchart"* page.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

32.1 Syntax for MCAxis.MoveHome

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.MoveHalt.Home.<Function or Access to Value> (optional
<Parameter 1, Parameter 2, Parameter n>);
```

32.2 Functions of MCAxis.MoveHome

This function offers the following functions:

– Start()
– SetReference()

Functions for reference shifting when using absolute encoders

– SetAbsEncoderReference()
– ClearAbsEncoderReferenceStatus()

32.2.1 Start()

This function initiates referencing for an MCX axis. Die Achse beginnt unmittelbar ihre Bewegung in die mit `Direction` vorgegebene Richtung. Their behavior when they reach the hardware limit switches depends on the type of referencing selected. For example, it can reverse on its own and continue moving in the opposite direction. The movement ends when one of the following occurs:

– The reference point has been found.
– A limit has been reached
– An error has been detected.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.
– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Directions	Directions (optional)	Defines the direction in which the referencing run is started (see Directions [▶ 156]).
SwitchConfig	ReferencingSwitchConfigs (optional)	Defines the switch configuration used for referencing (see SwitchConfig [▶ 157]).
Mode	Referencing Modes (optional)	Defines the type of reference point search (see Mode [▶ 158]).
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start ramp during positioning (see Acceleration [▶ 159]). Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.
SpeedSwitchSearch	Double (optional)	Defines the speed at which the axis moves to locate the limit and/or reference switch. (see SpeedSwitchSearch [▶ 159]).
MaxDistanceSwitchSearch	Double (optional)	Defines the maximum distance the axis may travel until it reaches the relevant limit switches during the referencing process. If this distance is exceeded, the referencing process is terminated with an error (see MaxDistanceSwitchSearch [▶ 160]).
SpeedReferenceSearch	Double (optional)	Defines the speed at which the axis searches for the reference point after the reference/limit switch has been detected (see SpeedReferenceSearch [▶ 161]).
MaxDistanceReferenceSearch	Double (optional)	Defines the maximum distance the axis is allowed to travel after the reference/limit switch has been detected until it reaches the reference point. If this distance is exceeded, the referencing process is terminated with an error (see MaxDistanceReferenceSearch [▶ 161]).
Reference Offset	Double (optional)	Starting position: Distance: Defines the number of position units by which the axis is moved after the reference is found, before the position is set to the home position (see Reference Offset [▶ 162]).
Normal Position	Double (optional)	Home Position; Defines the position to which the axis is moved after the referencing operation has been successfully completed (see Normal Position [▶ 163]).

Syntax

```
AX1.MoveHome.Start(Directions.Positive, ReferencingSwitchConfigs.RefAndLimitSwitch, ReferencingModes.ZeroPulse_2Phase,3000,500,200,20,15,125,0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	X		Working
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.2.2 SetReference()

This function sets the current position of the MCX axis to the value passed to the function.

After the controller restarts, this value is `0.0`. If no value is passed for the parameter, Motion Control uses the last value that was passed.

Parameters	data type	Meaning
Position	Double (optional)	Defines the reference position.

Syntax

```
AX1.MoveHome.SetReference(10.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.2.3 SetAbsEncoderReference()

ⓘ Info

Reference shift when using absolute encoders

This command is useful when absolute encoders are used for reference shifting.

For more information, refer to the JetSymonline help or the information sheet *titled "Saving and Retrieving the Reference Point for (Multiturn) Absolute Encoders (Saved Reference Offset)* ."

To use this command, the storage function must be enabled in NVRAM.

transfer parameter

The passed value is set to the parameter `Default value: Transferposition` (`NormalPosition`). If no value is provided, the last value specified in the `"Default"parameter is used : The value passedto the Position` (`NormalPosition`) property is used.

Unlike the "SetReference" function, the internally calculated reference offset is stored in the internal battery-backed RAM (and on any connected USB flash drive). When the controller is rebooted, this calculated value is used again for the reference offset, so that the actual position of the MCX axis can be controlled directly using the reference offset that was set previously. The reference status is then immediately set to "Initial position set."

Parameters	data type	Meaning
Position	Double (optional)	Write the parameter `"Default setting": Position` (`NormalPosition`).

Syntax

```
AX1.MoveHome.SetAbsEncoderReference(0.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.2.4 ClearAbsEncoderReferenceStatus()

ⓘ Info

Reference shift when using absolute encoders

This command is useful when absolute encoders are used for reference shifting.

For more information, refer to the JetSymonline help or the information sheet *titled "Saving and Retrieving the Reference Point for (Multiturn) Absolute Encoders (Saved Reference Offset)* ."

To use this command, the storage function must be enabled in NVRAM.

Use this function to clear the reference status of an axis. This has the following consequences:

– The reference state is set to `NoReference` .
– The `IsAbsEncoderReferenceValid` query returns `false` .
– The current reference offset remains in effect until the controller is restarted.

This function can also be used in applications without an absolute encoder to set the reference state to `NoReference` .

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.MoveHome.ClearAbsEncoderReferenceStatus();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3 Properties of MCAxis.MoveHome

This function offers the following properties:

– Directions
– SwitchConfig
– Mode
– Acceleration
– SpeedSwitchSearch

– MaxDistanceSwitchSearch
– SpeedReferenceSearch
– MaxDistanceReferenceSearch
– ReferenceOffset
– NormalPosition
– ReferenceState
– IsReferenceRunStarted
– IsSwitchFound
– IsReferenceFound
– IsReferenceSet

Functions for reference shifting when using absolute encoders

– IsAbsEncoderReferenceValid
– ReferenceShift

32.3.1 Directions

Description

data type	Meaning
Type Directions (optional)	Reads or writes the direction in which the referencing process is started.
Directions.positive	The axis starts the referencing run in the positive direction (counting direction of the actual position).
Directions.negative	The axis starts the referencing run in the negative direction (counting direction of the actual position).

Syntax

```
RefDirection := AX1.MoveHome.Direction;
AX1.MoveHome.Direction := Directions.Negative;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.2 SwitchConfig

Description

data type	Meaning
ReferencingSwitchConfigs (optional)	Reads or writes the switch configuration used for referencing.
ReferencingSwitchConfigs.NoSwitch	No switch is taken into account. The reference can only be set to the encoder's zero pulse. This setting can be used, for example, with modulo axes that do not have a gearbox.
ReferencingSwitchConfigs.RefAndLimitSwitch	Reference and limit switches are taken into account. Possible scenarios: – If the reference switch is reached before the limit switch, the reference is determined at the reference switch. – If the first limit switch is reached before the reference switch, the axis reverses direction. If the reference switch is then reached, the reference is determined at the reference switch. – If the second limit switch is reached before the reference switch, the referencing process is aborted due to an error.
ReferencingSwitchConfigs.RefSwitch	Only the reference switch is taken into account. If a limit switch is reached before the reference switch, the referencing process is aborted due to an error.
ReferencingSwitchConfigs.LimitSwitch	Only the limit switches are taken into account. The limit switches serve as reference switches.

Syntax

```
RefDirection := AX1.MoveHome.Direction;
AX1.MoveHome.Direction := Directions.Negative;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.3 Mode

Description

data type	Meaning
Referencing Modes (optional)	Reads or writes the mode used for referencing.
ReferencingModes.ZeroPulse_2Phase	The reference point is determined at the encoder's zero pulse. The referencing is performed in two phases.
Referencing-Modes.SwitchingEdge_2Phase	The reference point is determined at a switch edge (reference switch or limit switch). The referencing is performed in two phases.
ReferencingModes.ZeroPulse_1Phase	The reference point is determined at the encoder's zero pulse. The referencing operation is performed in a single phase.
Referencing-Modes.SwitchingEdge_1Phase	The reference point is determined at a switch edge (reference switch or limit switch). The referencing operation is performed in a single phase.

Syntax

```
RefMode := AX1.MoveHome.Mode;
AX1.MoveHome.Mode := ReferencingModes.SwitchingEdge_2Phase;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.4 Acceleration

Description

data type	Meaning
Double (optional)	Reads or writes the maximum acceleration to be achieved on the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Syntax

```
RefAcceleration := AX1.MoveHome.Acceleration;
AX1.MoveHome.Acceleration := 10000;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.5 SpeedSwitchSearch

Description

data type	Meaning
Double (optional)	Reads or writes the speed at which the axis moves to locate the limit and/or reference switch.

Syntax

```
RefSpeed1 := AX1.MoveHome.SpeedSwitchSearch;
AX1.MoveHome.SpeedSwitchSearch := 250;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.6 MaxDistanceSwitchSearch

Description

data type	Meaning
Double (optional)	Reads or writes the maximum distance the axis is allowed to travel before reaching the relevant switches for referencing. If this distance is exceeded, the referencing process is terminated with an error.

Syntax

```
MaxReferencePosition := AX1.MoveHome.MaxDistanceSwitchSearch;
AX1.MoveHome.MaxDistanceSwitchSearch := 5000.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.7 SpeedReferenceSearch

Description

data type	Meaning
Double (optional)	Reads or writes the speed at which the axis searches for the reference point after the reference/limit switch has been found. Referencing without a switch search and a single-phase referencing run are performed only at this speed.

Syntax

```
RefSpeed2 := AX1.MoveHome.SpeedReferenceSearch;
AX1.MoveHome.SpeedReferenceSearch := 50;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.8 MaxDistanceReferenceSearch

Description

data type	Meaning
Double (optional)	Reads or writes the maximum distance the axis is allowed to travel after the reference point switch has been detected. If this distance is exceeded, the referencing process is terminated with an error.

Syntax

```
MaxReferenceSearchPos := AX1.MoveHome.MaxDistanceReferenceSearch;
AX1.MoveHome.MaxDistanceReferenceSearch := 100.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.9 Reference Offset

Description

data type	Meaning
Double (optional)	Reads or writes the number of position units by which the axis is moved after the reference point is found, before the position is set to the home position.

Syntax

```
Home Position Distance := AX1.MoveHome.ReferenceOffset;
AX1.MoveHome.ReferenceOffset := 1265.5;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.10 NormalPosition

Description

data type	Meaning
Double (optional)	Reads or writes the position to which the axis is moved after the reference point has been found. The MCX system sets the target and actual positions of the axis to this value at the end of a successful referencing run.

Syntax

```
HomePosition := AX1.MoveHome.NormalPosition;
AX1.MoveHome.NormalPosition := 0.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.11 ReferenceState

Description

data type	Meaning
Type MCReferenceStates (optional)	Reads the status of the referencing run. This information allows you to determine whether the axis has already been homed, or at what point in the referencing process the axis is currently located. There are 5 reference run statuses that can be queried. Depending on the type of referencing, some of these statuses may be skipped or not accepted.
MCReferenceStates.NoReference	– The axis has not yet been referenced, meaning that successful referencing has not yet been performed or the `SetReference()` function has not yet been executed. – An error occurred on an axis with the `ReferenceSet` status while attempting to determine the reference again. – The ClearAbsEncoderReferenceStatus() [▶ 155] function was executed.

data type	Meaning
MCReferenceStates.Refer-enceRunStarted	Referencing has been initiated. The relevant limit or reference switch for the reference point has not yet been found.
MCReferenceStates.Switch-Found	The relevant limit or reference switch has been detected (transition from Phase 1 to Phase 2).
MCReferenceStates.Reference-Found	The reference mark has been found.
MCReferenceStates.Reference-Set	The foundation has been laid. The axis is now referenced, and the refer-encing process is complete.

Syntax

```
When AX1.MoveHome.ReferenceState == MCReferenceStates.ReferenceRunStarted ,
continue;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

32.3.12 IsReferenceRunStarted, IsSwitchFound, IsReferenceFound, IsReferenceSet

Description

data type	Meaning
Bool	Read the properties to determine whether the axis is in the specified reference state (see ReferenceState [▶ 163]). If none of these properties are met, the reference status of the axis is `NoReference`.

Syntax

```
Reference Run Start := AX1.MCAxis.MoveHome.IsReferenceRunStarted
```

state administration

The property can be queried in any state.

32.3.13 IsAbsEncoderReferenceValid

Description

> **ⓘ Info**
>
> **Reference shift when using absolute encoders**
>
> This command is useful when absolute encoders are used for reference shifting.
>
> For more information, refer to the JetSymonline help or the information sheet *titled "Saving and Retrieving the Reference Point for (Multiturn) Absolute Encoders (Saved Reference Offset)* ."
>
> To use this command, the storage function must be enabled in NVRAM.

data type	Meaning
Bool	Check whether the reference shift function is active for this MCX axis. If `True`, a reference shift is present (can also be 0) and is applied when the controller boots up.

Syntax

```
If AX1.MotionHome.IsAbsEncoderReferenceValid then …
```

state administration

The property can be queried in any state.

32.3.14 ReferenceShift

Description

> **ⓘ Info**
>
> **Reference shift when using absolute encoders**
>
> This command is useful when absolute encoders are used for reference shifting.
>
> For more information, refer to the JetSymonline help or the information sheet *titled "Saving and Retrieving the Reference Point for (Multiturn) Absolute Encoders (Saved Reference Offset)* ."
>
> To use this command, the storage function must be enabled in NVRAM.

data type	Meaning
Double (optional)	Reads or writes the number of position units by which the zero position of the MCX axis with an absolute encoder is offset from the encoder's zero point. This value can also be described. **NOTICE! However, this is only useful for very minor corrections**. We recommend using the `SetAbsEncoderReference()` function to set the position where the MCX axis should be located.

Syntax

```
Reference shift := AX1.MoveHome.ReferenceShift;
AX1.MoveHome.ReferenceShift := 257.5;
```

state administration

The property can be queried in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33 MCAxis.MovePtp

The **MCAxis.MovePtp** function allows you to initiate and configure point-to-point positioning for an MCX axis.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

33.1 Syntax for MCAxis.MovePtp

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.MovePtp.<Function or Access to Value>(optional <Parame-ter 1,
Parameter 2, Parameter n>);
```

33.2 Functions of MCAxis.MovePtp

This function offers the following functions:

- Start()
- StartNewTargetPosition()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()

33.2.1 Start()

This function initiates point-to-point positioning of an MCX axis. The axis begins its movement immediately or applies the data to the current position.

Since all the parameters required for positioning can be specified in a single command, this is the most concise way to fully configure and initiate a positioning movement.

transfer parameter

Please follow these rules:

- All parameters are specified in the units that the user has predefined for the axis.
- The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Positioning in buffered mode

In buffered mode, you can call up multiple positioning commands, which are executed one after the other. This means that the motion control unit initiates a new positioning operation as soon as the setpoint of a current positioning operation has reached the target. For more information on this topic, see the MCAxis.Buffer [▶ 32]section.

Positioning in unbuffered mode (default)

Depending on whether the axis is moving or not, the following behavior occurs:

- Calling the function in the **"Enabled"**state: The movement begins.
- Calling the function in the **running**state: The movement parameters are applied to the current OR subsequent movement according to the table below.

Parameters	data type	Meaning	Behavior when calling a function while in the "Running"state
Positioning Mode	Axis Positioning Modes (optional)	Defines the positioning mode [▶ 175].	The mode takes effect immediately.
TargetPosition	Double (optional)	Defines the target position at which the axis should stop after positioning is complete.	– If the new target position can be reached while adhering to the maximum values for speed, acceleration, deceleration, and jerk, the system moves directly to that position using the new motion parameters. – Otherwise, the current positioning operation will be completed, and the newly programmed one will be initiated immediately.
Speed	Double (optional)	Defines the speed at which the axis should be positioned when it is not in the ramp (acceleration, deceleration) phases. The value must be greater than `0`.	If a new positioning operation is initiated while the previous one is still in progress (Runningstate), the new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration ramp parameters.
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.	The speed ramps are implemented according to the new acceleration and deceleration values. – If the motion is currently in such a ramp, intermediate forms occur that do not correspond to the selected ramp type. – Transitions are always performed under the control of the jerk limiter.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop ramp while positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.	
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change to the target window is immediately applied to the current positioning.

Positioning phases

A positioning movement consists of the following phases:

1. acceleration ramp

2. Constant speed
   This phase occurs only if the speed can be reached using the specified MovePtp parameters. Otherwise, the acceleration ramp transitions directly into the deceleration ramp (triangular mode).
3. deceleration ramp

Types of ramps

The ramp profiles for the acceleration and deceleration phases are specified by the axis parameter `MCAxis.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (MCAxis.Mechanism.Slope [▶ 134]). There are two possible settings here:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration;
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = quadratic acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,20000,500,1000,1000,0.5);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis has been stopped (MCAxis.State [▶ 200]).

```
When Ax1.State.IsEnabled Continue; //The axis is in position control mode
and is stationary.
When Ax1.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **ramp status** indicates which phase of motion the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 134]).

```
When Ax1.Mechanism.Slope.IsAccelerating Continue;
When Ax1.Mechanism.Slope.IsAtConstantSpeed Continue;
When Ax1.Mechanism.Slope.IsDecelerating , continue;
When Ax1.Mechanism.Slope.IsStopped , continue;
```

– If you want to check whether a defined position has been reached shortly before the target, use the **target window** (MCAxis.IsInTargetWindow [▶ 111]).

```
If Ax1.IsInTargetWindow , Continue;
```

33.2.2 StartNewTargetPosition()

This function initiates point-to-point positioning of an MCX axis. The axis begins moving immediately or applies the data to adjust the current position.

This function is a simplified version of the `Start()` function, which passes only the relevant position parameters. The speed values and ramp settings for the axis are applied.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Positioning in buffered mode

In buffered mode, you can call up multiple positioning commands, which are executed one after the other. This means that the motion control unit initiates a new positioning operation as soon as the setpoint of the current positioning operation has reached the target. For more information on this topic, see the MCAxis.Buffer [▶ 32]section.

Positioning in unbuffered mode (default)

Depending on whether the axis is moving or not, the following behavior occurs:

– Calling the function in the **"Enabled"**state: The movement begins.

– Calling the function in the **running**state: The movement parameters are applied to the current OR subsequent movement according to the table below.

Parameters	data type	Meaning	Behavior when calling a function in the "Running"status
Positioning Mode	Axis Positioning Modes (optional)	Defines the positioning mode [▶ 175].	The mode takes effect immediately.
TargetPosition	Double (optional)	Defines the target position at which the axis should be positioned after the positioning operation has been completed.	– If the new target position can be reached while adhering to the maximum values for speed, acceleration, deceleration, and jerk, the system moves directly to that position using the new motion parameters. – Otherwise, the current positioning operation will be completed, and the newly programmed one will be initiated immediately.
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change to the target window is immediately applied to the current positioning.

Positioning phases

A positioning movement consists of the following phases:

1. Acceleration
2. Constant speed;

   This phase occurs only if the speed can be reached using the specified MovePtp parameters.
3. delay

Types of ramps

The ramp profiles for the acceleration and deceleration phases are specified by the axis parameter `MCAxis.Mechanism.Slope.SlopeType (type AxisSlopeTypes)` . There are two possible settings here:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration;
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration;
  deceleration phase = quadratic acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
AX1.MovePtp.StartNewTargetPosition(AxisPositioningModes.AbsNor-
mal,20000,0.5);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis has been stopped (MCAxis.State [▶ 200]).

```
When Ax1.State.IsEnabled Continue; //The axis is in position control mode
and is stationary.
When Ax1.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **ramp status** indicates which phase of motion the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 134]).

```
When Ax1.Mechanism.Slope.IsAccelerating Continue;
When Ax1.Mechanism.Slope.IsAtConstantSpeed Continue;
When Ax1.Mechanism.Slope.IsDecelerating , continue;
When Ax1.Mechanism.Slope.IsStopped , continue;
```

– If you want to check whether a defined position has been reached shortly before the target, use the **target window** (MCAxis.IsInTargetWindow [▶ 111]).

```
If Ax1.IsInTargetWindow , Continue;
```

33.2.3 NewSpeed()

The function sets a new value for the speed of a point-to-point positioning operation on an MCX axis. The following applies:

– The value must be greater than `0` .

– If the axis is already in motion (**running**state), the new speed value is applied immediately.

When an axis is in motion, the transition from the current speed to the new speed occurs at the acceleration rate set for the acceleration ramp. This is also the case when the speed decreases.

– If the axis is at a standstill (**enabled**state), the value is not applied until the next PtP positioning operation.

– The ramp type cannot be changed during positioning.

– Jerk limiting is active for all transitions if the ramp type is set to *sin* ².

ⓘ Info

Bring the axis to a stop

To bring an axis to a stop, use only the `MoveHalt` or `Stop` commands. It is **not** possible to stop an axis by setting its speed to `0` .

transfer parameter

Parameters	data type	Meaning	Behavior when calling a function while in the "running"state
Speed	Double	Defines the speed at which the axis should be positioned when it is not in the ramp (acceleration, deceleration) phase.	The new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk.

Syntax

```
AX1.MovePtp.NewSpeed(2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33.2.4 NewAcceleration()

The function sets a new value for the maximum acceleration of a point-to-point positioning operation on an MCX axis.

The following applies:

– An active acceleration ramp is not modified.

– The new value will be applied during the next PtP positioning.

transfer parameter

Parameters	data type	Meaning
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Syntax

```
AX1.MovePtp.NewAcceleration(2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33.2.5 NewDeceleration()

The function sets a new value for the maximum delay of a point-to-point positioning operation on an MCX axis.

The following applies:

– The value is also used during an ongoing PtP positioning if the target can still be reached with the new parameter.

– Otherwise, the value will be applied during the next PtP positioning.

transfer parameter

Parameters	data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop ramp while positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Syntax

```
AX1.MovePtp.NewDeceleration(2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted

Initial state	Allowed	Final state
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33.3 Properties of MCAxis.MovePtp

This function offers the following properties:

– Positioning Mode
– TargetPosition
– TargetWindow

33.3.1 Positioning Mode

Description

This property allows you to read the currently active positioning mode of an MCX axis or set a new value for that axis.

The following applies:

– A new value takes effect during the next point-to-point positioning, even if the change is passed during an ongoing positioning operation.
– After power-up, the final target position is calculated based on the actual position at the time of power-up.

data type	Meaning
Axis Positioning Mode	Mode for the type of PtP movement.
AxisPositioningMode.AbsNormal	Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement for positioning is determined by the last position and the new target position.
AxisPositioningMode.Relative-Target	Moves the axis by the distance specified in the `TargetPosition` parameter. The target position is calculated by adding the value in `TargetPosition`to the last target position. The direction of travel is determined by the sign of the `TargetPosition`parameter.
AxisPositioningMode.RelActual	Moves the axis by the distance specified in the `TargetPosition` parameter. The target position is calculated by adding the value in `TargetPosition`to the actual position at the start of the positioning process. The direction of travel is determined by the sign of the `TargetPosition`parameter.

data type	Meaning
AxisPositioningMode.AbsModuloPos	Applies only to modulo axes. Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement during positioning is always **positive**. In this case, the modulo wrap can be ignored if the new target position is less than the last target position.
AxisPositioningMode.AbsModuloNeg	Applies only to modulo axes. Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement during positioning is always **negative**. In this case, the modulo wrap can be ignored if the new target position is greater than the last target position.
AxisPositioningMode.AbsModuloAuto	Applies only to Modulo axes. – Positioning starts when the axis is paused: Moves the axis to the position specified in the `TargetPosition` parameter. It is approached **via the shortest path** relative to the reference position. The direction of movement for positioning is determined by the shortest path to the destination. – Positioning starts while the axis is moving: Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement for the positioning is determined by the direction of movement of the current positioning.
AxisPositioningMode.AbsModuloKeepDirection	Applies only to modulo axes. Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement for the positioning is determined by the direction of movement of the previous positioning.

Syntax

```
AX_PositioningMode := AX1.MovePtp.PositioningMode;
AX1.MovePtp.PositioningMode := AxisPositioningModes.RelTarget;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33.3.2 TargetPosition

Description

This property allows you to read the current value of `TargetPosition` for an MCX axis or write a new value for that axis. This will take effect when the next point-to-point positioning begins.

data type	Meaning
Double	Property used to determine the target position based on the `PositioningMode`property.

Syntax

```
Target := AX1.MovePtp.TargetPosition;
AX1.MovePtp.TargetPosition := 12000;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33.3.3 TargetWindow

Description

This property allows you to read the currently valid value of the target window setting for an MCX axis or write a new value for that axis. This will take effect when the next point-to-point positioning begins.

data type	Meaning
Double	Defines when the `IsInTargetWindow` status becomes active in the target position area of a positioning operation.

Syntax

```
TargetWindow := AX1.MovePtp.TargetWindow;
AX1.MovePtp.TargetWindow := 0.7;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

33.4 Example task MCAxis.MovePtp

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    run: bool; //Auxiliary flag for advancing in the program
End_Var;

Task Sample2 Autorun

loop
    when run Continue;

    //Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled Continue;

    AX1.MoveHome.SetReference(0.0); //Set reference position to start
position
    When AX1.MoveHome.IsReferenceSet Continue;

    //Position the axis absolutely at 10,000 mm - Maximum speed 500 mm/s -
    //Max. Acceleration 1000 mm/s² - Max. deceleration 1000 mm/s²
    //Target window 0.1 mm
    AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,10000,500,1000,5000,0.1
);
    When AX1.IsInTargetWindow Continue;

    //When the axis has reached its position, wait 1 second.
    Delay(t#1s);

    //Reposition the axis back along the same path; here, relative to the
last target position.
    AX1.MovePtp.StartNewTargetPosition(AxisPositioningModes.RelTarget,-10000
);
    when AX1.IsInTargetWindow Continue;
```

```
    //When the axis has reached its position, wait 1 second.
    Delay(t#1s);

 //If the user has since cleared the auxiliary flag in the setup, lock the
axis and wait for it to be set again.
    //Otherwise, repeat the axis reversal.
    If Not run Then
        AX1.Power.Disable();
        When AX1.State.IsDisabled Continue;
    End_if;

end_loop;

End_Task;
```

34 MCAxis.MoveVelocity

The **MCAxis.MoveVelocity** function allows you to start and configure the continuous positioning of an MCX axis.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

34.1 Syntax for MCAxis.MoveVelocity

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.MoveVelocity.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

34.2 Functions of MCAxis.MoveVelocity

This function offers the following functions:

– Start()
– Reverse()
– NewSpeed()
– NewAcceleration()
– NewDeceleration()

34.2.1 Start()

This function initiates continuous positioning of an MCX axis. The axis begins its movement immediately or applies the data to the current position.

The axis accelerates at the current or programmed ramp rate and continues to move at the current or programmed `speed` until it is stopped by a hold or stop command. Continuous positioning is position-controlled, so that a position tracking error is also monitored during movement.

Since all parameters required for positioning can be specified in a single command, this is the most concise way to fully configure and initiate a positioning movement.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.
– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Positioning in buffered mode

In buffered mode, you can call up multiple positioning commands, which are executed one after the other. This means that the motion control unit initiates a new positioning operation as soon as the setpoint of a current positioning operation has reached the target. For more information on this topic, see the MCAxis.Buffer [▶ 32]section.

Positioning in unbuffered mode (default)

Depending on whether the axis is moving or not, the following behavior occurs:

– Calling the function in the **"Enabled"**state: The movement begins.

– Calling the function in the **running**state: The movement parameters are applied to the current OR subsequent movement according to the table below.

Parameters	data type	Meaning	Behavior when calling a function while in the "Running"state
Directions	Directions (optional)	Defines the direction of movement (see Directions [▶ 187]).	The new direction parameters are applied immediately. If the new direction is opposite to the current direction of motion, a reversal operation is performed using the acceleration ramps.
Speed	Double (optional)	Defines the speed at which the axis should be positioned when it is not in the ramp (acceleration, deceleration) phases. The value must be greater than `0` .	If a new positioning operation is initiated while the previous one is still in progress (**Running**state), the new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration ramp parameters.
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.	The speed ramps are implemented according to the new acceleration and deceleration values. – If the motion is currently in such a ramp, intermediate forms occur that do not correspond to the selected ramp type. – Transitions are always performed under the control of the jerk limiter.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the next `MoveHalt.Stop()` command on the stop ramp, unless a different value is specified there. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.	

Types of ramps

The ramp profiles for the acceleration and deceleration phases are specified by the axis parameter `MCAxis.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (see MCAxis.Mechanism.Slope [▶ 134]). There are two possible settings here:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration;
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration;
  deceleration phase = quadratic acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
AX1.MoveVelocity.Start(Directions.Positive,5000,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Working
Working	√	Working	Working
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis has been stopped (MCAxis.State [▶ 200]).

```
When Ax1.State.IsEnabled Continue; //The axis is in position control mode
and is stationary.
When Ax1.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **ramp status** indicates which phase of motion the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 134]).

```
When Ax1.Mechanism.Slope.IsAccelerating Continue;
When Ax1.Mechanism.Slope.IsAtConstantSpeed Continue;
When Ax1.Mechanism.Slope.IsDecelerating , continue;
When Ax1.Mechanism.Slope.IsStopped , continue;
```

34.2.2 Reverse()

This function reverses the direction of a continuous motion and moves the axis in the new direction at the last set speed. The last acceleration ramp set is used to reverse the direction of motion.

ⓘ Info

Usage of the `Reverse()` function

– This command cannot be applied to a point-to-point movement (`MovePtp`) that is currently in progress.

– If the command is issued while the axis is in the **enabled** state, no action is taken.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.MoveVelocity.Reverse();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	X		Blocked
Approved	–		Approved
Working	√	Working	Working
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

34.2.3 NewSpeed()

The function sets a new speed value for continuous positioning on an MCX axis. The following applies:

– The value must be greater than `0` .

– If the axis is already in motion (**running**state), the new speed value is applied immediately.

The transition from the current speed to the new speed for a running axis is performed using the values set for the acceleration ramps. This is also the case when the speed decreases.

– If the axis is at a standstill (in the **enabled**state), the value is not applied until the next positioning operation.
– The ramp type cannot be changed during positioning.
– Jerk limiting is active for all transitions if the ramp type is set to *sin ²*.

ⓘ Info

Bring the axis to a stop

To bring an axis to a stop, use only the `MoveHalt` or `Stop` commands. It is **not** possible to stop an axis by setting its speed to `0` .

transfer parameter

Parameters	data type	Meaning	Behavior when calling a function while in the "Running"state
Speed	Double	Defines the speed at which the axis should be positioned when it is not in the ramp (acceleration, deceleration) phase.	If a new positioning operation is initiated while the previous one is still in progress (Runningstate), the new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration ramp parameters.

Syntax

```
AX1.MoveVelocity.NewSpeed(2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.
– a state in which calling the function is not recommended, but does not cause an error.
X the state in which calling the function is not permitted and results in an error.

34.2.4 NewAcceleration()

The function sets a new value for the maximum acceleration of an infinite positioning cycle on an MCX axis.

The following applies:

– An active acceleration ramp is not modified.
– The new value will be applied during the next continuous positioning operation.

transfer parameter

Parameters	data type	Meaning
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Syntax

```
AX1.MoveVelocity.NewAcceleration(2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

34.2.5 NewDeceleration()

The function sets a new value for the maximum delay of an infinite positioning operation on a MCX axis.

The value is used during the next `MoveHalt.Start()` command or the next positioning operation.

transfer parameter

Parameters	data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop ramp while positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Syntax

```
AX1.MoveVelocity.NewDeceleration(2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

34.3 Properties of MCAxis.MoveVelocity

This function offers the following properties:

– Directions

34.3.1 Directions

Description

Use the `Direction` property to read the current direction of an MCX axis or set a new value for that axis. This will take effect the next time continuous positioning is used. This is also the case if the change is made while positioning is in progress.

data type	Meaning
Directions (optional)	Direction of continuous motion.
Directions.Positive	The axis is moved in the positive direction.
Directions.Negative	The axis is moved in the negative direction.

Syntax

```
AX_Direction := AX1.MoveVelocity.Direction;
AX1.MoveVelocity.Direction := Directions.Negative;
```

state administration

The property can be queried in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved

Initial state	Allowed	Final state
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

34.4 Example task MCAxis.MoveVelocity

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    run: bool; //Auxiliary flag for advancing in the program
    Direction_positive: bool;
End_Var;

Task Sample 4 : Autorun

Direction_positive := true;

loop
    //The "run" flag acts as a prompt button. As long as it is true, the
axis moves
When run , continue;

    // Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled , continue;

    //The "Direction_positive" flag is used to specify the direction of
movement
    If Direction_positive Then
        AX1.MoveVelocity.Start(Directions.Positive,100,1000,5000);
    Else
        AX1.MoveVelocity.Start(Directions.Negative,100,1000,5000);
    End_If;

    //If the program is to be paused (run = false), call MoveHalt
    If not run , continue;
        AX1.MoveHalt.Start(MCAxisHaltModes.Normal,);
    //If the axis is at a standstill, turn it off again;
    When AX1.State.IsEnabled , continue;
    AX1.Power.Disable();
    When AX1.State.IsDisabled , continue;
end_loop;

End_Task;
```

35 MCAxis.Position

The **MCAxis.Position** function allows you to access position information for an MCX axis.

35.1 Syntax for MCAxis.Position

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Position.<Access to Value>;
```

35.2 Functions of MCAxis.Position

This function offers the following functions:

Functions for MCX axes with absolute encoders

– AbsEncOverflowHandler.Activate()
– AbsEncOverflowHandler.Deactivate()

35.2.1 AbsEncOverflowHandler.Activate()

ⓘ Info

MCX axes with absolute encoders

For MCX axes equipped with absolute encoders, an overflow handling function is available; the functions and properties of the Motion API 2.x described here are intended for use with this function.

In this context, please refer to the information in the JetSym Help or the separate information sheets *on consistent operation of an axis with a multiturn absolute encoder (with a non-binary gear) in the event of encoder position overflow*.

Please note that the storage function must be enabled in NVRAM in order to use these commands.

Use the `AbsEncOverflowHandler.Activate()` function to enable overflow handling for the MCX axis with an absolute encoder.

Normally, this function is activated only once during commissioning of the axis. This setting is configured in the axis's motion setup on the "Referencing" page. Therefore, this command is only needed if the function must be enabled later using the STX program.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.Position.AbsEncOverflowHandler.Activate();
```

state administration

The function can be called from any state.

35.2.2 AbsEncOverflowHandler.Deactivate()

ⓘ Info

MCX axes with absolute encoders

For MCX axes equipped with absolute encoders, an overflow handling function is available; the functions and properties of the Motion API 2.x described here are intended for use with this function.

In this context, please refer to the information in the JetSym Help or the separate information sheets *on consistent operation of an axis with a multiturn absolute encoder (with a non-binary gear) in the event of encoder position overflow*.

Please note that the storage function must be enabled in NVRAM in order to use these commands.

Use the `AbsEncOverflowHandler.Deactivate()` function to disable overflow handling for the MCX axis with an absolute encoder.

Normally, there is no need to disable this feature unless it was accidentally enabled. This setting is configured in the axis's motion setup on the "Referencing" page.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.Position.AbsEncOverflowHandler.Deactivate();
```

state administration

The function can be called from any state.

35.3 Properties of MCAxis.Position

This function offers the following properties:

General Properties for MCX Axes

- Target
- Actual
- Setpoint
- Encoder Position
- ActualTrackingError
- TrackingErrorLimit
- TrailingIndicatorMax
- TrailingIndicatorMin

Properties specifically for MCX axes with absolute encoders

- AbsEncOverflowHandler.Deactivate() [▶ 190]
- AbsEncOverflowHandler.Activate() [▶ 189]
- AbsEncOverflowHandler.IsActive

35.3.1 Target

Description

data type	Meaning
Double	Reads the target position of the MCX axis [in the user-defined position unit]. The following applies: – If the axis is in motion, this is the position at which the positioning is completed. – If the axis is at a standstill or in an error state, this is the target position of the last positioning command executed.

Syntax

```
TargetPosition := AX1.Position.Target;
```

state administration

The property can be read in any state.

35.3.2 Actual

Description

data type	Meaning
Double	Reads the current position of the MCX axis (actual position) [in the user-defined position unit].

Syntax

```
CurrentPosition := AX1.Position.Actual;
```

state administration

The property can be read in any state.

35.3.3 Setpoint

Description

data type	Meaning
Double	Reads the currently calculated target position of the MCX axis [in the user position unit]. The target position is recalculated by the MCX in each MCX cycle based on the positioning data.

Syntax

```
TargetPosition := AX1.Position.Setpoint;
```

state administration

The property can be read in any state.

35.3.4 Encoder Position

Description

data type	Meaning
Double	Reads the position output by the MCX axis's position sensor [in the user-defined position unit]. This value differs from the actual position due to offsets that are calculated during referencing or, if applicable, during modulo overflow. Output in user position units, taking the mechanical ratio into account.

Syntax

```
EncoderRawValue := AX1.Position.EncoderPosition;
```

state administration

The property can be read in any state.

35.3.5 CurrentTrackingError

Description

data type	Meaning
Double	Reads the value for the current tracking error [in the user position unit]. The tracking error is calculated in the drive by subtracting the setpoint position from the actual position in each cycle of the position controller. It is a measure of how well the drive can follow the MCX's setpoint specifications.

Syntax

```
Tracking error := AX1.Position.ActualTrackingError;
```

state administration

The property can be read in any state.

35.3.6 TrailingIndicatorMax

Description

data type	Meaning
Double	Reads the highest positive tracking error detected by the control system since the measurement was last reset [in the user position unit]. The trailing indicator measurement can be used to detect slow changes in the trailing error during repetitive movements (see also `AX1.Drive.Control.Position.ResetTrailingIndicators()`).

Syntax

```
MaxPosTrailingError := AX1.Position.TrailingIndicatorMax;
```

state administration

The property can be read in any state.

35.3.7 TrailingIndicatorMin

Description

data type	Meaning
Double	Reads the highest negative tracking error detected by the control system since the measurement was last reset [in the user position unit]. The trailing indicator measurement can be used to detect slow changes in the trailing error during repetitive movements (see also `AX1.Drive.Control.Position.ResetTrailingIndicators()`).

Syntax

```
MinNegTrailingError := AX1.Position.TrailingIndicatorMin;
```

state administration

The property can be read in any state.

35.3.8 AbsEncOverflowHandler.IsActive

ⓘ Info

MCX axes with absolute encoders

For MCX axes equipped with absolute encoders, an overflow handling function is available; the functions and properties of the Motion API 2.x described here are intended for use with this function.

In this context, please refer to the information in the JetSym Help or the separate information sheets *on consistent operation of an axis with a multiturn absolute encoder (with a non-binary gear) in the event of encoder position overflow*.

Please note that the storage function must be enabled in NVRAM in order to use these commands.

transfer parameter

data type	Meaning
Bool	Check whether the overflow handling function has been enabled for this MCX axis.

Syntax

```
If AX1.Position.AbsEncOverflowHandler.IsActive then …
```

state administration

The property can be read in any state.

36 MCAxis.Power

The **MCAxis.Power** function allows you to enable or disable the control—and thus the torque output—of an MCX axis.

36.1 Syntax for MCAxis.Power

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Power.<Function>(optional <Parameter>;
```

36.2 Features of MCAxis.Power

This function offers the following functions:

– Enable()
– Disable()
– Quickstop()

36.2.1 Enable()

This function activates the servo amplifier's controllers and applies voltage to the motor. When power is applied, the drive is able to generate torque at the motor shaft. If the control system is properly configured and the torque is sufficient, the axis will be held at its current position because the target position value takes precedence over the actual position value.

Approval Requirements

A real axis can only be enabled if the servo amplifier is operational. Check the state of the servo amplifier as follows:

```
If AX1.Drive.IsReadyForOperation Then
```

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.Power.Enable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Approved
Approved	–	Approved
Working	–	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Therefore, check whether the new state has been reached in the program before issuing any further commands for the axis.

```
When Ax1.State.IsEnabled , continue;
```

36.2.2 Disable()

This function disables the servo amplifier for torque generation (pulse inhibit). The motor will immediately lose power and coast to a stop if no brakes are applied.

Approval Requirements

A real axis can only be enabled if the servo amplifier is operational. Check the state of the servo amplifier as follows:

```
If AX1.Drive.IsReadyForOperation Then
```

transfer parameter

Parameters	data type	Meaning
DisableModes	MCPowerDisableMode (optional)	Mode for executing the `Disable()` function
	MCPowerDisableMode.Forced (Default)	The pulse block takes effect immediately in all cases. This mode is also active if no parameter is specified.
	MCPowerDisableMode.StateCheck	The pulse inhibit function is only active when the axis is in the *Enabled* operating state. In all other states, an error message is displayed.

Syntax

```
Ax1.Power.Disable();
Ax1.Power.Disable(MCPowerDisableModes.Forced);
Ax1.Power.Disable(MCPowerDisableModes.StateCheck);
```

state administration

The following table applies when writing the function:

The following table applies when the function is called without parameters or with the parameter `MCPowerDisableModes.Forced`:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	√	Blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The following table applies to calls using the `MCPowerDisableModes.StateCheck` parameter:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before issuing any further commands for the axis, check whether the new state has been reached in the program as follows:

```
If Ax1.State.IsDisabled , continue;
```

If the axis was previously in the *"Error: Enabled"* state, use the following command. The position buffers are being cleared.

```
When AX1.State.IsDisabledError , continue;
```

36.2.3 QuickStop()

This function stops the axis using the quick-stop delay and locks the servo amplifier once a standstill has been reached.

Approval Requirements

A real axis can only be enabled if the servo amplifier is operational. Check the state of the servo amplifier as follows:

```
If AX1.Drive.IsReadyForOperation Then
```

transfer parameter

The function has no transfer parameters.

Syntax

```
Ax1.Power.QuickStop();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	√	Blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before issuing any further commands for the axis, check whether the new state has been reached in the program as follows:

```
If Ax1.State.IsDisabled , continue;
```

36.3 Example task: MCAxis.Power

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample 1 : Autorun
    //Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled , continue;

    //This section can contain a process that controls the axis or other
tasks that move the axis
    //…

    // Lock axis
    If AX1.State.IsRunning Then    // if the axis is still moving
        AX1.Power.QuickStop();    // Stop using the quick stop slope
    Else
        AX1.Power.Disable();    // otherwise lock immediately
    End_if;
    When AX1.State.IsDisabled , continue;

End_Task;
```

37 MCAxis.Settings

The **MCAxis.Settings** function allows you to adjust or save all parameters of an MCX axis at once.

Do not adjust parameters while a save operation is in progress

Saving the parameters to a file is a process that runs in the background. In other words, it spans several cycles of the MCX. If parameter values are changed during saving, it is impossible to predict whether the old or new value will be written to the file.

How Parameter Management Works

Parameters saved in JetSym 's motion setup are stored in a text file on the PC and, if requested, also in the controller's file system (Motionsubfolder. The filename of the parameter file on the controller is *MCAxis_X.ini*, where *X* stands for the axis ID number. The axis ID is displayed on the Axis Parameters page in the MCX axis's motion setup. In addition, the tooltip in the Hardware tree displays the axis ID when the mouse pointer hovers over the axis.

When the controller boots up, the MCX kernel reads the data from the *MCAxis_X.ini*file and uses it for the relevant axis. If such a file is present on the controller, no further parameterization is required. If no corresponding file exists for the axis, the parameters are loaded with default values.

Normally, no modifications to the file are required from within the STX program.

In very rare cases, however, it may be advisable to reload the parameter file onto the controller or to save the parameters. However, when saving, it is important to note that the parameter file saved in the JetSym project may no longer match the parameters currently in effect on the machine.

37.1 Syntax for MCAxis.Settings

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.Settings.<Function>(optional <Parameter>;
```

37.2 Features of MCAxis.Settings

This function offers the following functions:

– SaveToFile()
– LoadFromFile()

37.2.1 SaveToFile()

transfer parameter

Parameters	data type	Meaning
Suffix	Integer (optional)	Enables saving of the MCX axis parameters to a file in the "Motion" subfolder within the controller's file system. Depending on the transfer parameter, different files are created or overwritten – Parameter `0` or no parameter: The parameter file *MCaxis_X.ini* for this axis is created or overwritten with the current values. – Parameter not equal to `0`: The parameter file *MCAxis_X_TmpY.ini* is created or overwritten with the current values. In this case , *Y* is equal to the parameter `suffix`.

Syntax

```
AX1.Settings.SaveToFile();     // Overwrites (or creates) the file
MCAxis_1.ini (if AX1 has axis ID 1)
AX1.Settings.SaveToFile(3);     // Creates the file MCAxis_1_Tmp3.ini (if AX1
has axis ID 1)
```

state administration

The function can be called from any state.

37.2.2 LoadFromFile()

transfer parameter

Parameters	data type	Description
Suffix	Integer (optional)	Enables the import of MCX axis parameters from a file located in the "Motion" subfolder within the controller's file system. Depending on the transfer parameter, different files are read. – Parameter `0` or no parameter: The data from the *MCaxis_X.ini* file is read and used for the relevant axis. – Parameter not equal to 0: The data from the *MCaxis_TmpY.ini* file is read and used for the relevant axis. In this case , *Y* is equal to the parameter `suffix`.

Syntax

```
AX1.Settings.LoadFromFile();     //Reads the MCAxis_1.ini file (if AX1 has
axis ID 1)
AX1.Settings.LoadFromFile(3);     //Reads the file MCAxis_1_Tmp3.ini (if AX1
has axis ID 1)
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

38 MCAxis.State

The **MCAxis.State** function allows you to retrieve various pieces of information about an MCX axis, such as operating state, motion type, etc.

Basic and advanced information

In this context, please also note the following information:

– For technology groups, the MCAxis.State.Techno [▶ 212]section;
– For state transitions, see the MCAxis.State.Transitions [▶ 219]section;
– For a detailed description of the MCX core architecture, see the JetSym online help
– For basic information, see Motion Control Application Note No. 54 (MCX Motion Control Basics).

operating state

The operating state (`OperatingMode`) specifies the basic operating principle of the MCX axis. There are five operating states:

operating state	Meaning
`Inactive`	The axis cannot accept commands, for example, because it is being used by a group.
`Normal` (normal)	The axis is usable. Traverse commands are executed in a non-blocking manner (see MCAxis.Buffer [▶ 32]).
`Buffered`	The axis is usable. Traverse commands are buffered and executed (see MCAxis.Buffer [▶ 32]).
`External`	The axis is an external axis. Only the current position can be read. It can only be used as a master axis within a technology group.
`Error`	An operating error has occurred. The operating state is retained; however, no new functions or parameters can be passed.

Tab. 2: Operating states of MCX axes

Operating status

At any given time, an MCX axis is in one, and only one, clearly defined operating state (`OperatingState`). For example, if the axis is in the **"Running"** state, the **"Enabled"** state is not active, even though it does, of course, have a pulse enable signal.

In a given state, only a subset of all function calls is permitted. The state model for the axis is as follows:

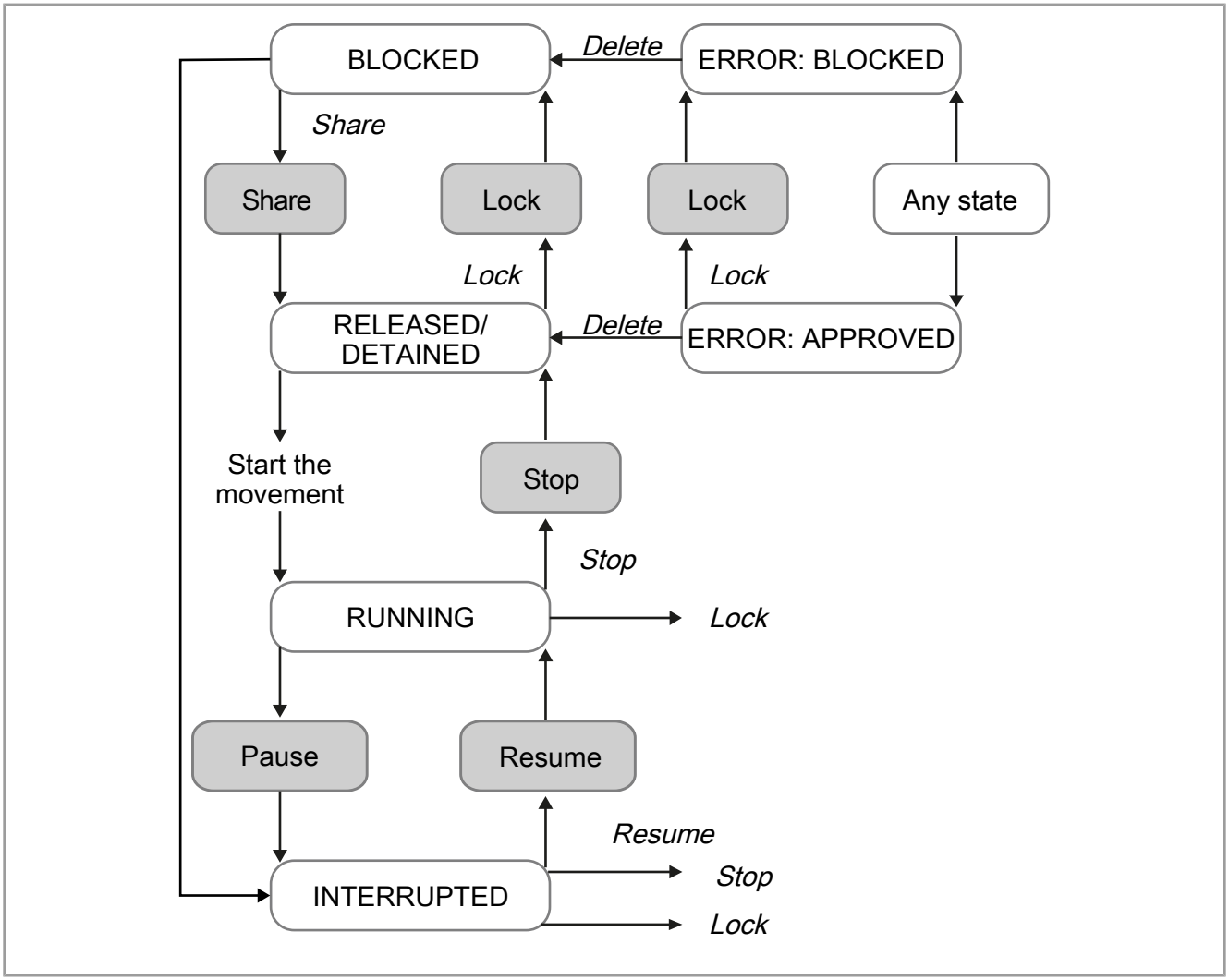


Fig. 3: State model for MCX axes

Operating status	Meaning
Inactive	The axis cannot accept commands, for example because it is being used by a group.
Disabled	– The servo amplifier's control circuits are open (no pulse enable). – No torque is generated.
Enabled	– The controllers are closed (pulse enable). – The setpoint interpolator is at a standstill (the lead-in buffer is empty).
Running (In progress)	The axis is in motion; that is, the setpoint interpolator transmits motion setpoints to the servo amplifier.
Suspended	– This corresponds to the **"Enabled"** operating state, with the exception that all pre-fill buffers remain filled but are locked. A motion that has been paused using `MCAxis.State.Transitions.Suspend()` can be resumed using the `MCAxis.State.Transitions.Resume()` command. – In buffered mode, it is also possible to pause the processing of the Buffer (`MCAxis.Buffer.Suspend()`) and resume it (`MCAxis.Buffer.Resume()`).
EnabledError (Error: enabled)	An error has occurred. After the error response, the axis remains in control mode (error level 4).

Operating status	Meaning
DisabledError (Error: Locked)	An error has occurred. After the error response, the axis disables control (error levels 5 and 6).

Tab. 3: Operating states of MCX axes

Type of movement

The motion type (`TypeOfMotion`) specifies the type of motion that the setpoint generator is currently generating for the MCX axis.

Querying the transaction type is useful, for example, in a monitoring task where the process is unknown. Based on the active motion type prior to shutdown, it is possible to control the restart correctly.

Operating status	Meaning
None (`MoveNone`)	The setpoint generator is out of service because no movement is requested.
Positioning (`MovePtp`)	A positioning operation (with a programmed target) is being performed.
Continuous motion (`MoveVelocity`)	An endless positioning operation is performed: Start with an acceleration ramp, then drive at a programmed speed.
Referencing (`MoveHome`)	Referencing is being performed.
Active coupling (`MoveCoupling`) [1)]	The axis is a secondary axis in a technology group. The coupling process is being performed.
Coupling (`MoveCoupled`) [1)]	The axis is a secondary axis in a technology group. It follows the leading axis in accordance with the programmed linkage movement (electric gear, cam disc)
Decoupling active (`MoveDecoupling`) [1)]	The axis is a secondary axis in a technology group. The disengagement process is being performed.
Decoupled (`MoveDecoupled`) [1)]	The axis is a secondary axis in a technology group. It is not connected to the drive shaft.
Positioning, superimposed (`MovePtpSuperPose`) [1)]	The axis is a secondary axis in a technology group. It is in a superimposed position (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).
Continuous motion, superimposed (`MoveVelocitySuperPose`) [1)]	The axis is a secondary axis in a technology group. It is in a superimposed continuous motion (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).

[1)] It is recommended that these queries be executed via the technology group in which the MCX axis is active as a secondary axis.

38.1 Syntax for MCAxis.State

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.State.<Access to Value>
```

38.2 Properties of MCAxis.State

This function offers the following properties:

- Value
- IsDisabled
- IsEnabled
- IsRunning
- IsSuspended
- IsDisabledError
- IsEnabledError
- IsInactive
- Operating Mode
- TypeOfMotion

38.2.1 Value

Description

data type	Meaning
OperatingStates	Reads the current operating state of the axis (for a detailed description, see the *"Operating State"* section in MCAxis.State [▶ 200]).
OperatingStates.Inactive	Inactive
OperatingStates.Disabled	Blocked
OperatingStates.Enabled	Approved
OperatingStates.Running	Working
OperatingStates.Suspended	Interrupted
OperatingStates.EnabledError	Error: Approved
OperatingStates.DisabledError	Error: blocked

Syntax

```
OpState := AX1.State.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.2 IsDisabled

Description

data type	Meaning
Bool	Check whether the axis is in the **"Locked"** operating state.

Syntax

```
When AX1.State.IsDisabled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.3 IsEnabled

Description

data type	Meaning
Bool	Check whether the axis is in the **"Enabled"** state.

Syntax

```
When AX1.State.IsEnabled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.4 IsRunning

Description

data type	Meaning
Bool	Check whether the axis is in the **"Running"** state.

Syntax

```
When AX1.State.IsRunning , Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.5 IsSuspended

Description

data type	Meaning
Bool	Check whether the axis is in the **"Interrupted"** operating state.

Syntax

```
When AX1.State.IsSuspended , Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.6 IsDisabledError

Description

data type	Meaning
Bool	Check whether the axis is in the **"Error: Locked"** state.

Syntax

```
When AX1.State.IsDisabledError , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.7 IsEnabledError

Description

data type	Meaning
Bool	Check whether the axis is in the **"Error: Enabled"** state.

Syntax

```
When AX1.State.IsEnabledError , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.8 IsInactive

Description

data type	Meaning
Bool	Check whether the axis is in the **"Inactive"** state.

Syntax

```
When AX1.State.IsInactive , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.9 OperatingMode.Value

Description

data type	Meaning
Type MCOperatingModes	Reads the operating state in which the axis is currently set (for a detailed description, see the *"Operating State"* section in MCAxis.State [▶ 200]).

data type	Meaning
MCOperatingModes.Inactive	The axis is inactive, perhaps because it is managed by a group.
MCOperatingModes.Normal	The axis is in non-buffered mode. This is the most commonly used mode.
MCOperatingModes.Buffered	The axis is in buffered mode. This operating state is normally used only by axes in a path group.
MCOperatingModes.External	The axis is an external axis. It simply evaluates the current position of the second encoder on a real (internal) axis.
MCOperatingModes.Error	The axis has encountered an operating error.

Syntax

```
OpModeAxis := AX1.State.OperatingMode.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInInactiveMode

Description

data type	Meaning
Bool	Check whether the axis is in the *inactive* operating state.

Syntax

```
When AX1.State.OperatingMode.IsInInactiveMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInNormalMode

Description

data type	Meaning
Bool	Check whether the axis is in the *Normal* operating state.

Syntax

```
If AX1.State.OperatingMode.IsInNormalMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInBufferedMode

Description

data type	Meaning
Bool	Check whether the axis is in the *buffered* operating state.

Syntax

```
If AX1.State.OperatingMode.IsInBufferedMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInExternalMode

Description

data type	Meaning
Bool	Check whether the axis is in the *external* operating state.

Syntax

```
When AX1.State.OperatingMode.IsInExternalMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInErrorMode

Description

data type	Meaning
Bool	Check whether the axis is in the *error* operating state.

Syntax

```
If AX1.State.OperatingMode.IsInErrorMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.2.10 TypeOfMotion.Value

Description

data type	Meaning
Type MCAxisTypesOfMotion	Reads the motion mode in which the axis is currently operating (for a detailed description, see the *"Motion Mode"* section in MCAxis.State [▶ 200]).
MCAxisTypesOfMotion.MoveNone	None
MCAxisTypesOfMotion.MovePtp	Positioning
MCAxisTypesOfMotion.MoveVelocity	Continuous motion
MCAxisTypesOfMotion.MoveHome	referencing
MCAxisTypesOfMotion.MoveCoupling	Coupling engagement
MCAxisTypesOfMotion.MoveCoupled	Coupling
MCAxisTypesOfMotion.MoveDecoupling	Disengage active
MCAxisTypesOfMotion.MoveDecoupled	Decoupled
MCAxisTypesOfMotion.MovePtpSuperPose	Positioning (superimposed)
MCAxisTypesOfMotion.MoveVelocitySuperPose	Continuous motion (superimposed)

Syntax

```
TypeOfMotion := AX1.State. TypeOfMotion.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveNone

Description

data type	Meaning
Bool	Check whether the axis is in the "None" movement type.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveNone , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMovePtp

Description

data type	Meaning
Bool	Checks whether the axis is in the "Positioning" motion mode.

Syntax

```
When AX1.State.TypeOfMotion.IsMovePtp , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveVelocity

Description

data type	Meaning
Bool	Checks whether the axis is in the "Endless Motion" motion type.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveVelocity , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveHome

Description

data type	Meaning
Bool	Check whether the axis is in the "referencing" motion mode.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveHome , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveCoupling

Description

data type	Meaning
Bool	Check whether the axis is in the "coupling" motion mode.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveCoupling , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveCoupled

Description

data type	Meaning
Bool	Check whether the axis is in the "Coupling" motion type.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveCoupled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveDecoupling

Description

data type	Meaning
Bool	Check whether the axis is in the "Decoupling" motion mode.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveDecoupling , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveDecoupled

Description

data type	Meaning
Bool	Check whether the axis is in the "Decoupled" motion mode.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveDecoupled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMovePtpSuperPose

Description

data type	Meaning
Bool	Checks whether the axis is in the "Positioning" (superimposed) motion mode.

Syntax

```
When AX1.State.TypeOfMotion.IsMovePtpSuperPose , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveVelocitySuperPose

Description

data type	Meaning
Bool	Check whether the axis is in the "Infinite motion (superimposed)" motion type.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveVelocitySuperPose , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

38.3 Example task MCAxis.State

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample2 Autorun

    //Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled , continue;

    //This section can contain a process involving the axis, or other tasks
that move the axis
    //…

    // Lock axis
    If AX1.State.IsRunning Then //if the axis is still moving
        AX1.Power.QuickStop(); //Stop using the quick stop slope
    Else
        AX1.Power.Disable(); //otherwise disable immediately
    End if;
    When AX1.State.IsDisabled , continue
End_Task;
```

39 MCAxis.State.Techno

The **MCAxis.State.Techno** function allows you to read the status of the coupling for an MCX axis that is the slave axis in a technology group (`techno`).

Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

39.1 Syntax for MCAxis.State.Techno

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.State.Techno.<Access to Value>;
```

39.2 Properties of MCAxis.State.Techno

This function offers the following properties:

- TechnoID
- SyncState
- IsDecoupled
- IsDecoupling
- IsCoupling
- IsCoupled
- SyncPosition
- Cam.CamID
- Cam.CamSegmentID
- Cam.MasterOffset
- Cam.FollowerOffset
- Cam.MasterFactor
- Cam.FollowerFactor

39.2.1 TechnoID

Description

data type	Meaning
Integer	Reads the number of the active Techno group to which the MCX axis is connected. If the Techno group is not active, the value `0` is returned.

Syntax

```
ActiveTechno := AX1.State.Techno.TechnoID;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.2 SyncState

Description

data type	Meaning
MCSyncStates	Reads the coupling status of the MCX axis in the active technology group. If the MCX axis is not part of any active technology group, " `Decoupled` " is returned. If the MCX axis is the reference axis in an active technology group, " `Coupled`" is always returned.
MCSyncStates.Decoupled	The MCX axis is decoupled.
MCSyncStates.Decoupling	The MCX axis is currently being decoupled.
MCSyncStates.Coupling	The MCX axis is currently being synchronized.
MCSyncStates.Coupled	The MCX axis is coupled.

Syntax

```
If AX1.State.Techno.SyncState == MCSyncStates.Coupled then …
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.3 IsDecoupled

Description

data type	Meaning
Bool	Checks whether the MCX axis is decoupled in an active technology group. If the MCX axis is not part of any active technology group, `IsDecoupled` is also `true`.

Syntax

```
If AX1.State.Techno.IsDecoupled then …
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.4 IsDecoupling

Description

data type	Meaning
Bool	Checks whether the MCX axis in an active technology group is in the decoupling process.

Syntax

```
If AX1.State.Techno.IsDecoupling then …
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.5 IsCoupling

Description

data type	Meaning
Bool	Checks whether the MCX axis is in the coupling process within an active technology group.

Syntax

```
If AX1.State.Techno.IsCoupling then …
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.6 IsCoupled

Description

data type	Meaning
Bool	Checks whether the MCX axis is in an active technology group. If the MCX axis is the reference axis in an active technology group, `IsCoupled` is also `true`.

Syntax

```
If AX1.State.Techno.IsCoupled then …
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.7 SyncPosition

Description

data type	Meaning
Double	Reads the position where a decoupled slave axis would need to be in order to be ready for coupling without any offset. Querying this property is only relevant when the system is in a decoupled state. This allows you to determine which follower axis position corresponds to the current master axis position, so that you can, for example, move the follower axis to the synchronized position where coupling can occur without misalignment. If the MCX axis is subject to coupling or acts as the master axis, the `SyncPosition` corresponds to the target position of the slave axis.

Syntax

```
Sync_Pos := AX1.State.Techno.SyncPosition;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.8 Cam.CamID

ⓘ Info

Further information

For information on how a cam disc works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property includes several additional properties that allow you to retrieve information about an active cam disc involving the MCX axis.

These queries are only relevant for axes that are sequential axes within a technology group:

– If the MCX axis is not associated with any active cam disc, a value `of 0` is returned for all read operations on these properties.

– Even though the MCX axis is the primary axis in the technology group, a value `of 0` is returned for all read operations on these properties.

Description

data type	Meaning
Integer	Reads the number of the active cam disc involved in the MCX axis. If the technology group in question is not active, the value `0` is returned.

Syntax

```
ActiveCAM := AX1.State.Techno.Cam.CamID;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.9 Cam.CamSegmentID

ⓘ Info

Further information

For information on how a cam disc works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property includes several additional properties that allow you to retrieve information about an active cam disc involving the MCX axis.

These queries are only relevant for axes that are sequential axes within a technology group:

– If the MCX axis is not associated with any active cam disc, a value `of 0` is returned for all read operations on these properties.

– Even though the MCX axis is the primary axis in the technology group, a value `of 0` is returned for all read operations on these properties.

Description

data type	Meaning
Integer	Reads the number of the active segment of the cam disc that the MCX axis is engaged with. If the technology group is not active, the value `0` is returned.

Syntax

```
ActiveCamSegment := AX1.State.Techno.Cam.CamSegmentID;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.10 Cam.MasterOffset

ⓘ Info

Further information

For information on how a cam disc works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property includes several additional properties that allow you to retrieve information about an active cam disc involving the MCX axis.

These queries are only relevant for axes that are sequential axes within a technology group:

– If the MCX axis is not associated with any active cam disc, a value `of 0` is returned for all read operations on these properties.

– Even though the MCX axis is the primary axis in the technology group, a value `of 0` is returned for all read operations on these properties.

Description

data type	Meaning
Double	Reads the offset of the reference axis that was passed when the cam disc coupling was initiated for the active cam disc involving the MCX axis. If the technology group or cam disc is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveGuideOffset := AX1.State.Techno.Cam.MasterOffset;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.11 Cam.FollowerOffset

ⓘ Info

Further information

For information on how a cam disc works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property includes several additional properties that allow you to retrieve information about an active cam disc involving the MCX axis.

These queries are only relevant for axes that are sequential axes within a technology group:

– If the MCX axis is not associated with any active cam disc, a value `of 0` is returned for all read operations on these properties.
– Even though the MCX axis is the primary axis in the technology group, a value `of 0` is returned for all read operations on these properties.

Description

data type	Meaning
Double	Reads the offset of the secondary axis that was passed when the cam coupling was started for the active cam disc in which the MCX axis is involved. If the technology group or cam disc is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveFollowerOffset := AX1.State.Techno.Cam.FollowerOffset;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.12 Cam.MasterFactor

ⓘ Info

Further information

For information on how a cam disc works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property includes several additional properties that allow you to retrieve information about an active cam disc involving the MCX axis.

These queries are only relevant for axes that are sequential axes within a technology group:

– If the MCX axis is not associated with any active cam disc, a value `of 0` is returned for all read operations on these properties.
– Even though the MCX axis is the primary axis in the technology group, a value `of 0` is returned for all read operations on these properties.

Description

data type	Meaning
Double	Reads the lead axis factor that was passed when the cam coupling was initiated for the active cam disc in which the MCX axis is involved. If the technology group or cam disc is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveControlFactor := AX1.State.Techno.Cam.MasterFactor;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

39.2.13 Cam.FollowerFactor

ⓘ Info

Further information

For information on how a cam disc works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property includes several additional properties that allow you to retrieve information about an active cam disc involving the MCX axis.

These queries are only relevant for axes that are sequential axes within a technology group:

– If the MCX axis is not associated with any active cam disc, a value `of 0` is returned for all read operations on these properties.

– Even though the MCX axis is the primary axis in the technology group, a value `of 0` is returned for all read operations on these properties.

Description

data type	Meaning
Double	Reads the factor of the secondary axis that was passed when the cam coupling was initiated for the active cam disc in which the MCX axis is involved. If the technology group or cam disc is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveFollowFactor := AX1.State.Techno.Cam.FollowerFactor;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

40 MCAxis.State.Transitions

The **MCAxis.State.Transitions** function allows you to control state transitions of an MCX axis for specific applications.

As a rule, state transitions for an MCX axis are triggered automatically by motion commands and the behavior of the axis movement. Direct control of state administration is required only in exceptional cases—such as when responding to an error. The **MCAxis.State.Transitions**function, as well as `Diagnostics.ClearErrors()`, can be used to explicitly trigger state transitions.

ⓘ Info

Effective Date of the Orders

Even in buffered mode, the functions and properties of **MCAxis.State.Transitions** take effect immediately. All buffer tanks are emptied.

The **MCAxis.State** function allows you to retrieve various pieces of information about an MCX axis, such as operating state, motion type, etc.

Basic and additional information

In this context, please also note the following information:

– For technology groups, the MCAxis.State [▶ 200]section;
– For state transitions, see the MCAxis.State.Techno [▶ 212]section;
– For a detailed description of the MCX core architecture, see the JetSym online help
– For basic information, see Motion Control Application Note No. 54 (MCX Motion Control Basics).

operating state

The operating state (`OperatingMode`) specifies the basic operating principle of the MCX axis. There are five operating states:

operating state	Meaning
`Inactive`	The axis cannot accept commands, for example, because it is being used by a group.
`Normal` (normal)	The axis is usable. Traverse commands are executed in a non-blocking manner (see MCAxis.Buffer [▶ 32]).
`Buffered`	The axis is usable. Traverse commands are buffered and executed (see MCAxis.Buffer [▶ 32]).
`External`	The axis is an external axis. Only the current position can be read. It can only be used as a master axis within a technology group.
`Error`	An operating error has occurred. The operating state is retained; however, no new functions or parameters can be passed.

Tab. 4: Operating states of MCX axes

Operating status

At any given time, an MCX axis is in one, and only one, clearly defined operating state (`OperatingState`). For example, if the axis is in the **"Running"**state, the **"Enabled"** state is not active, even though it does, of course, have a pulse enable signal.

In a given state, only a subset of all function calls is permitted. The state model for the axis is as follows:

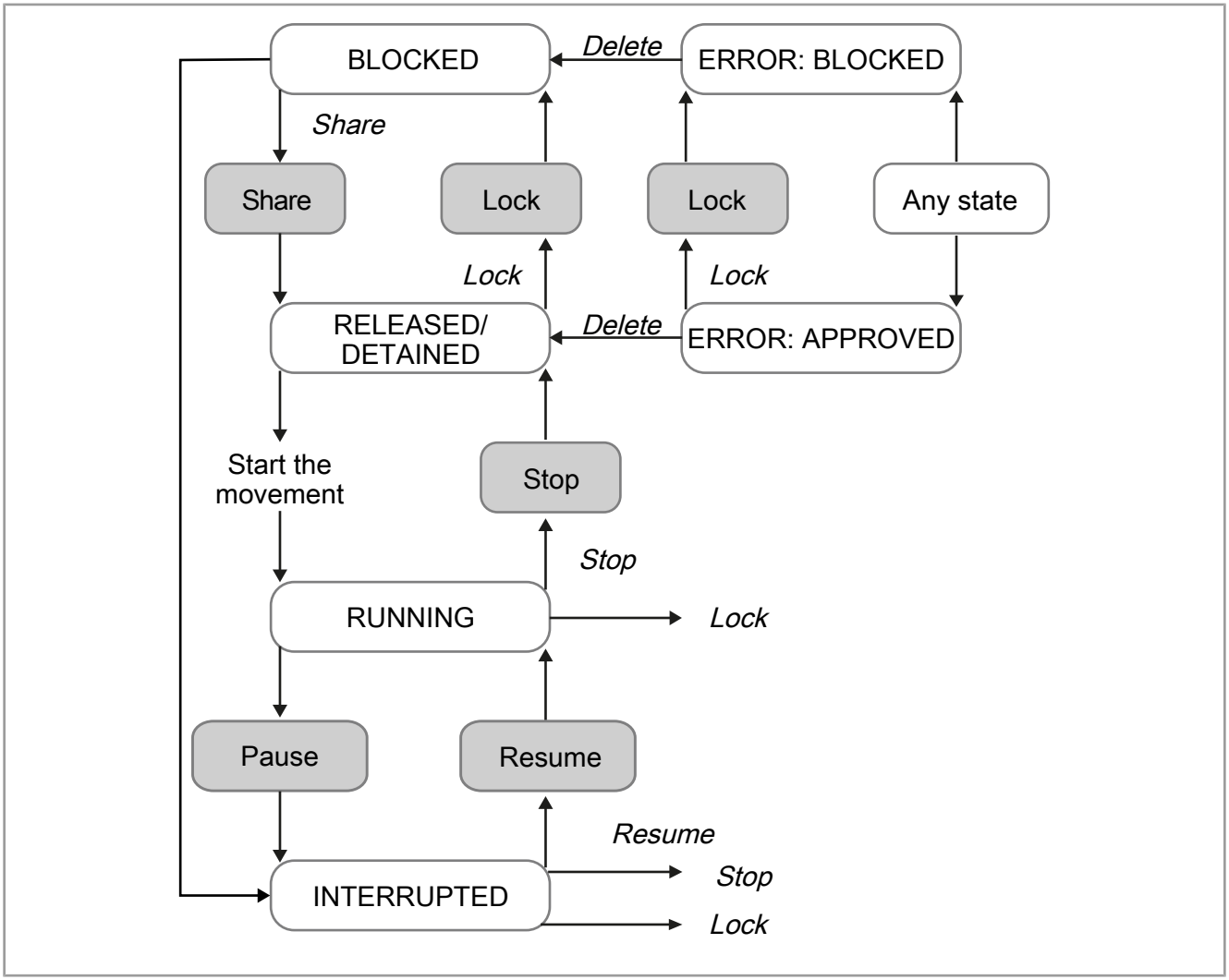


Fig. 4: State model for MCX axes

Operating status	Meaning
Inactive	The axis cannot accept commands, for example because it is being used by a group.
Disabled	– The servo amplifier's control circuits are open (no pulse enable). – No torque is generated.
Enabled	– The controllers are closed (pulse enable). – The setpoint interpolator is at a standstill (the lead-in buffer is empty).
Running (In progress)	The axis is in motion; that is, the setpoint interpolator transmits motion setpoints to the servo amplifier.
Suspended	– This corresponds to the **"Enabled"** operating state, with the exception that all pre-fill buffers remain filled but are locked. A motion that has been paused using `MCAxis.State.Transitions.Suspend()` can be resumed using the `MCAxis.State.Transitions.Resume()` command. – In buffered mode, it is also possible to pause the processing of the Buffer (`MCAxis.Buffer.Suspend()`) and resume it (`MCAxis.Buffer.Resume()`).

Operating status	Meaning
EnabledError (Error: enabled)	An error has occurred. After the error response, the axis remains in control mode (error level 4).
DisabledError (Error: Locked)	An error has occurred. After the error response, the axis disables control (error levels 5 and 6).

Tab. 5: Operating states of MCX axes

Type of movement

The motion type (`TypeOfMotion`) specifies the type of motion that the setpoint generator is currently generating for the MCX axis.

Querying the transaction type is useful, for example, in a monitoring task where the process is unknown. Based on the active motion type prior to shutdown, it is possible to control the restart correctly.

Operating status	Meaning
None (`MoveNone`)	The setpoint generator is out of service because no movement is requested.
Positioning (`MovePtp`)	A positioning operation (with a programmed target) is being performed.
Continuous motion (`MoveVelocity`)	An endless positioning operation is performed: Start with an acceleration ramp, then drive at a programmed speed.
Referencing (`MoveHome`)	Referencing is being performed.
Active coupling (`MoveCoupling`) [1)]	The axis is a secondary axis in a technology group. The coupling process is being performed.
Coupling (`MoveCoupled`) [1)]	The axis is a secondary axis in a technology group. It follows the leading axis in accordance with the programmed linkage movement (electric gear, cam disc)
Decoupling active (`MoveDecoupling`) [1)]	The axis is a secondary axis in a technology group. The disengagement process is being performed.
Decoupled (`MoveDecoupled`) [1)]	The axis is a secondary axis in a technology group. It is not connected to the drive shaft.
Positioning, superimposed (`MovePtpSuperPose`) [1)]	The axis is a secondary axis in a technology group. It is in a superimposed position (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).
Continuous motion, superimposed (`MoveVelocitySuperPose`) [1)]	The axis is a secondary axis in a technology group. It is in a superimposed continuous motion (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).

[1)] It is recommended that these queries be executed via the technology group in which the MCX axis is active as a secondary axis.

40.1 Syntax for MCAxis.State.Transitions

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.State.Transitions.<Function>(optional <Parameter>);
```

40.2 Functions of MCAxis.State.Transitions

This function offers the following functions:

– Enable()
– Disable()
– ErrorDisable()
– ErrorStop()
– ErrorLockUserInput()
– Suspend()
– Resume()
– UserPositionReset()

40.2.1 Enable()

The `State.Transitions.Enable()` function is identical to the `MCAxis.Power.Enable()` function. For a detailed description of this function, see the MCAxis.Power.Enable() [▶ 194]section.

Syntax

```
AX1.State.Transitions.Enable();
```

40.2.2 Disable()

Depending on the parameter passed (see the table below), this function performs the same action as the commands `MCAxis.Power.Disable()` and `MCAxis.Power.Quickstop()`. For more details, please refer to the descriptions for MCAxis.Power [▶ 194].

transfer parameter

Parameters	data type	Meaning
Mode	MCStateTransitionDisableModes	Mode for executing the `Disable()` function
	MCStateTransitionDisable-Modes.ImmediatePulseBlocking	Performs the same function as the `MCAxis.Power.Disable()` command or `MCAxis.Power.Disable(MCPowerDisable-Modes.Forced)`
	MCStateTransitionDisable-Modes.VelocityControlSlope	Performs the same function as the `AX1.Power.QuickStop()` command.

Syntax

```
AX1.State.Transitions.Disable(MCStateTransitionDisableModes.ImmediatePulse-
Blocking);
AX1.State.Transitions.Disable(MCStateTransitionDisableModes.VelocityControlS
lope);
```

40.2.3 ErrorDisable()

This function triggers an error in the MCX axis, which results in the **"Error: Locked"** state. The error response can be configured (see the ErrorDisableMode [▶ 227]property). In buffered mode, the pre-buffer is cleared.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.State.Transitions.ErrorDisable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: blocked
Approved	√	Error: blocked
Working	√	Error: blocked
Interrupted	√	Error: blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

40.2.4 ErrorStop()

This function triggers an error in the MCX axis, which results in the **"Error: Enabled"** state. The error response can be configured (see the ErrorStopMode [▶ 227]property). In buffered mode, the pre-buffer is cleared.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.State.Transitions.ErrorStop();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: Approved
Approved	√	Error: Approved
Working	√	Error: Approved
Interrupted	√	Error: Approved
Error: Approved	-	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

40.2.5 ErrorLockUserInput()

This function clears an operator error in the MCX axis. In buffered mode, the pre-run buffer is not cleared.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.State.Transitions.ErrorLockUserInput();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

40.2.6 Suspend()

Use this function to interrupt a positioning operation that is currently in progress. Deceleration occurs either using the last programmed deceleration slope or the configured maximum deceleration. The state after stopping is not *"Released"* but *"Paused."* An axis in this state can be moved to its original target position using the `Resume()` function.

If the MCX axis cannot be stopped before the programmed target position using the specified deceleration, the command is ignored and the positioning operation is completed. The state is then *"Released"* and not *"Paused."* In buffered mode, the run-ahead buffer is not cleared.

transfer parameter

Parameters	data type	Meaning
Mode	MCStateTransitionStopModes	Mode for the deceleration slope of the `Suspend()` function
	MCStateTransitionStopModes.ProgrammedDeceleration	Uses the last deceleration value programmed for the MCX axis.
	MCStateTransitionStopModes.MaxDeceleration	Uses the maximum deceleration configured for the MCX axis.

Syntax

```
AX1.State.Transitions.Suspend(MCStateTransitionStopModes.ProgDeceleration);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	X	Approved
Working	√	Interrupted
Interrupted	–	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

40.2.7 Resume()

This function resumes positioning that was paused by the Suspend() [▶ 224] function. The system moves to the originally specified target position. The speed and acceleration values are also carried over from the original position.

The `Resume()` function only takes effect if the MCX axis is in the *"Paused"* state.

In buffered mode, the output buffer is not cleared.

transfer parameter

The function has no transfer parameters.

Syntax

```
AX1.State.Transitions.Resume();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	X	Approved
Working	–	Working
Interrupted	√	Working
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

40.2.8 UserPositionReset()

This function performs a special operation to reset the setpoint generator. **Calling this function is only useful if the axis is no longer moving.**

Executing this function cancels any ongoing positioning and overwrites the target value of the MCX axis position with the actual value. This causes the position tracking error and the velocity setpoint to drop to zero immediately, bringing the moving axis to a stop at once.

A typical use case is referencing a stop. In this case, with reduced torque limitation, the motor is positioned to a location that is safely beyond the limit switch. If the axis stops moving, the `UserPositionReset()` function is executed to bring the setpoint generator to a stop at the limit position. The reference position is then set using `SetReference()` .

The `UserPositionReset()` function is also used after the MCAxis.Drive.TorqueShutOff [▶ 102] function has completed.

Pass-by-reference parameters

The function has no transfer parameters.

Syntax

```
AX1.State.Transitions.Resume();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Approved
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

40.3 Properties of MCAxis.State.Transitions

This function offers the following properties:

– ErrorDisableMode

– ErrorStopMode

40.3.1 ErrorDisableMode

Description

data type	Meaning
MCStateTransitionDisableModes	Error response used when the `ErrorDisable()` function is called.
MCStateTransitionDisableModes.VelocityControlSlope	Stops the axis with the quick-stop delay. Once a standstill is reached, the servo amplifier is disabled.
MCStateTransitionDisableModes.ImmediatePulseBlocking	Immediately disables the servo amplifier from generating torque (pulse inhibit).

Syntax

```
AX1.State.Transitions.ErrorDisableMode := MCStateTransitionDisableModes.ImmediatePulseBlocking;
ErrorResponse := AX1.State.Transitions.ErrorDisableMode;
```

state administration

The property can be read and written in any state.

When the property is read, the currently valid error response for the command is returned.

40.3.2 ErrorStopMode

Description

data type	Meaning
MCStateTransitionErrorStopModes	Error response used in the `ErrorStop()` function
MCStateTransitionErrorStopModes.ProgrammedDeceleration	Stops the axis with the programmed deceleration.
MCStateTransitionErrorStopModes.MaxDeceleration	Stops the axis with maximum deceleration.
MCStateTransitionErrorStopModes.AtBlockEnd	In buffered mode: Stops the axis at the end of the stroke.

Syntax

```
AX1.State.Transitions.ErrorStopMode := MCStateTransitionStopModes.MaxDeceleration;
ErrorResponse := AX1.State.Transitions.ErrorStopMode;
```

state administration

The property can be read and written in any state.

41 MCAxis.Stop

The **MCAxis.Stop** function allows you to stop an MCX axis that is in motion using a defined ramp.

Unlike the `MoveHalt.Start()` function, `Stop` clears all pre-fill buffers.

The function has no functions or properties.

> **ⓘ Info**
>
> **Further information**
>
> Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

41.1 Features of MCAxis.Stop

`MCAxis.Stop` is a method of the axis object.

41.1.1 Stop()

transfer parameter

Please note the following:

- This function executes the stop command immediately. The parameter passed (see table) specifies the maximum delay for this.
- If no parameter is passed, the most recently passed `StopMode` is used.
- This parameter can also be modified using the commands `MCAxis.State.Transitions.Suspend(..)` or `MCAxis.State.Transitions.Stop(..)` .

Parameters	data type	Meaning
StopModes	Enum StopModes (optional)	There are two delay values available.
	StopModes.ProgDeceleration	The deceleration last specified for the axis is used for stopping (default).
	StopModes.MaxDeceleration	To stop, the maximum acceleration of the axis is used as deceleration. This was either specified during commissioning in the motion setup under "Maximum Deceleration" or programmed using the command AX1.Mechanism.Limitations.MaxAcceleration.

Syntax

```
AX1.Stop(StopModes.ProgDeceleration);
AX1.Stop(StopModes.MaxDeceleration);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	X		Blocked
Approved	√		Approved

Initial state	Allowed	During execution	Final state
Working	√	Working	Approved
Interrupted	√		Approved
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning.

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis is stopped.

```
When Ax1.State.IsEnabled Continue; //The axis is in position control mode
and is stationary.
When Ax1.State.IsRunning Continue; //The setpoint generator generates a
motion.
```

– The **slope status** indicates which phase of the motion cycle the setpoint generator is currently in.

```
When Ax1.Mechanism.Slope.IsAccelerating Continue;
When Ax1.Mechanism.Slope.IsAtConstantSpeed Continue;
When Ax1.Mechanism.Slope.IsDecelerating , continue;
When Ax1.Mechanism.Slope.IsStopped , continue;
```

– This function does not affect the **target window status** .

41.2 Example task: MCAxis.Stop

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    run: bool; //Auxiliary flag for advancing in the program
End_Var;

Task Sample2 Autorun

loop
    When run , continue;

    // Turning on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled , continue;

    AX1.MoveHome.SetReference(0.0); //Set the reference position to the
power-on position
    When AX1.MoveHome.IsReferenceSet , continue;

    //Position the axis ("Automatic Sequence")
    AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,1000,
Speed1,1000,5000,0.1);
```

```
    //If the user has since cleared the auxiliary flag in the setup, move
the axis at the programmed maximum deceleration
    //Stop and lock
    //This event could be, for example, pressing the emergency stop button
    If not run , continue;
    AX1.Stop(StopModes.MaxDeceleration);
    when AX1.State.IsEnabled Continue;
    AX1.Power.Disable();
    When AX1.State.IsDisabled , continue;

end_loop;

End_Task;
```

42 MCAxis.TouchProbe

The **MCAxis.TouchProbe** function allows you to quickly save the position of the MCX axis based on a sensor signal.

Typical use cases:

– Position corrections based on sensor signals (e.g., registration mark correction)

– Position detection (e.g., irregularly distributed products on a conveyor belt)

– Product length measurement

For this function, the servo amplifier provides one or two digital inputs, depending on the device family, which enable a particularly fast response. Since up to two Touch Probe channels are available, this function is implemented as an array. In this array, only the values 1 (first touch probe channel) and 2 (second touch probe channel) are valid.

Applications with two touch probe channels

If you only need one channel on a servo amplifier with two Touch Probe channels, you should always use the first channel, as the internal processing for the first channel is faster than for the second.

The position can be saved based on the rising or falling edge of the sensor signal. It is also possible to evaluate both edges. In this case, the position is saved twice: once on the rising edge and once on the falling edge.

Single Mode and Continuous Mode

In **single mode** , the position detection function is initiated from within the STX program, and the stored position is processed before a new detection is initiated. In most cases, this is the recommended setting.

In single mode, a position window can be specified to define the range within which the sensor signal is to be accepted. If the sensor signal is detected outside the position window, it is ignored. If the entire position window is scanned (in either the positive or negative direction) and no signal is detected within the position window, the status “noEventInWindow” is set.

In **Continuous Mode** , however, the function remains active at all times, and the stored position is overwritten with every sensor signal.

Restrictions

– For a standard axis, the value for the upper position window limit must always be greater than the value for the lower position window limit.

– If, for a modulo axis, the value for the lower position window limit is greater than the value for the upper position window limit, the position window is centered on the origin (modulo wrap).

– The position window limits must be within or identical to the limits of the MCX axis's travel range. If the window function is not to be used, it is advisable to set the position window limits to match the travel range limits.

– *Single mode*only: When using the *"Both edges"* configuration, the status does not change from *"wait"* to *"both found*" until both edges have been detected. No intermediate state *such as "rising edge found"* or *"falling edge found"* is entered.

– Position window monitoring is not available in *Continuous Mode* .

– In *Continuous Mode* , the "*bothedges*" configuration is not applicable.

Sample flowcharts for various configurations

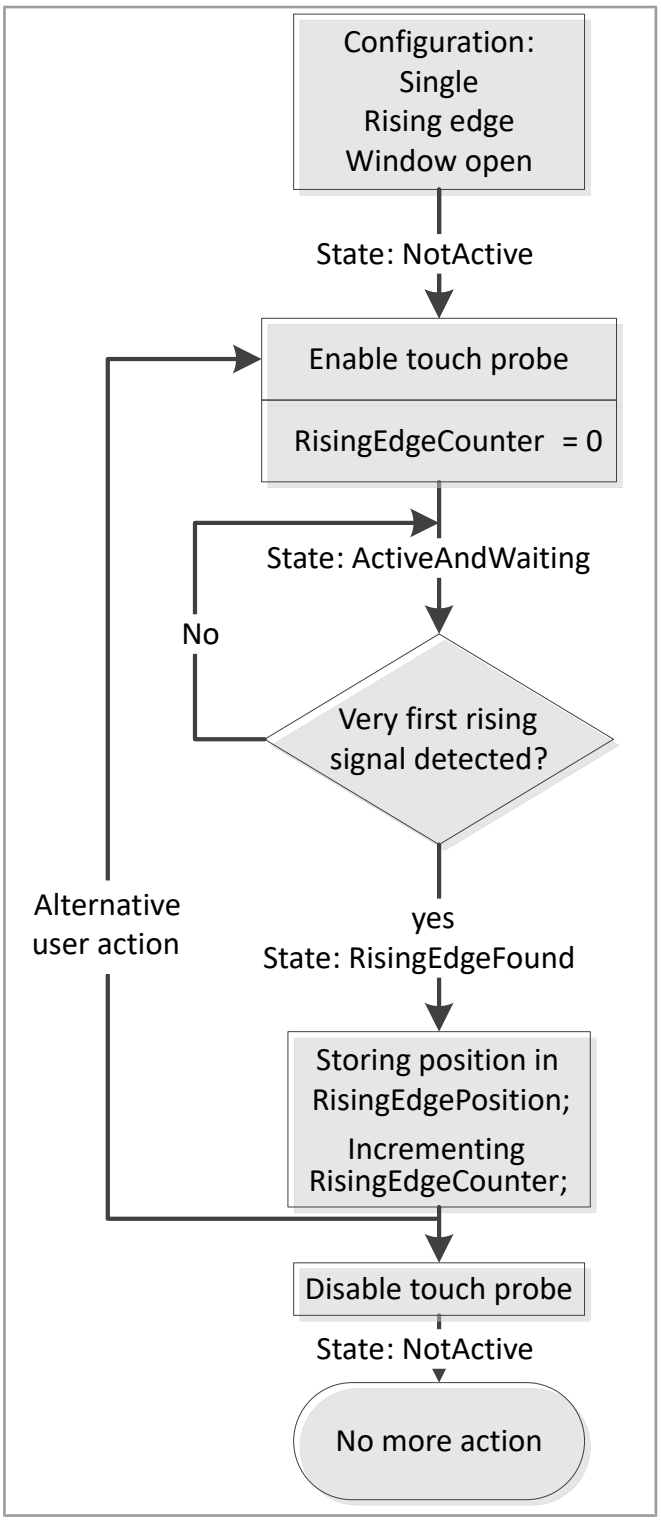


Fig. 5: *Flowchart for Touch Probe Configuration: Single Mode, Rising Edge, Open Position Window*

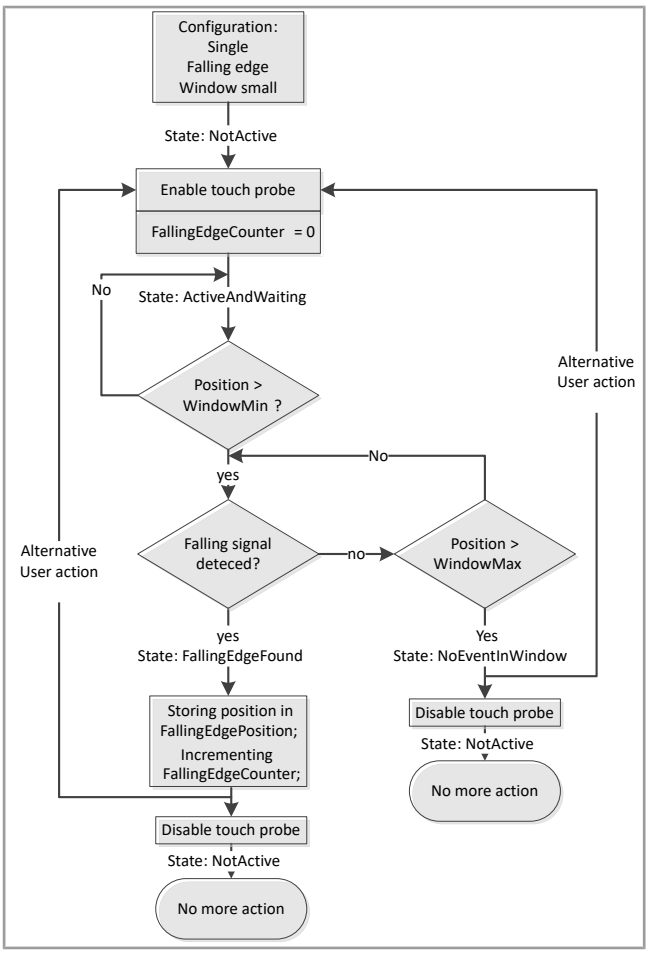


Fig. 6: *Flowchart for Touch Probe Configuration: Single Mode, Falling Edge, Small Position Window*

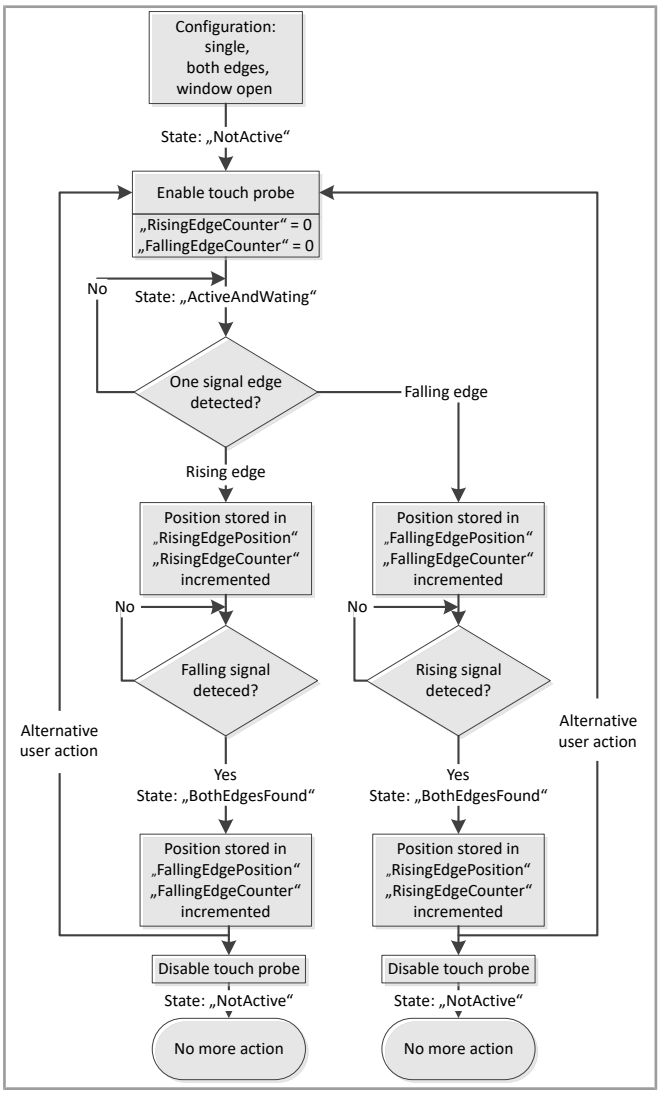


Fig. 7: *Flowchart for Touch Probe Configuration: Single Mode, Both Edges, Open Position Window*

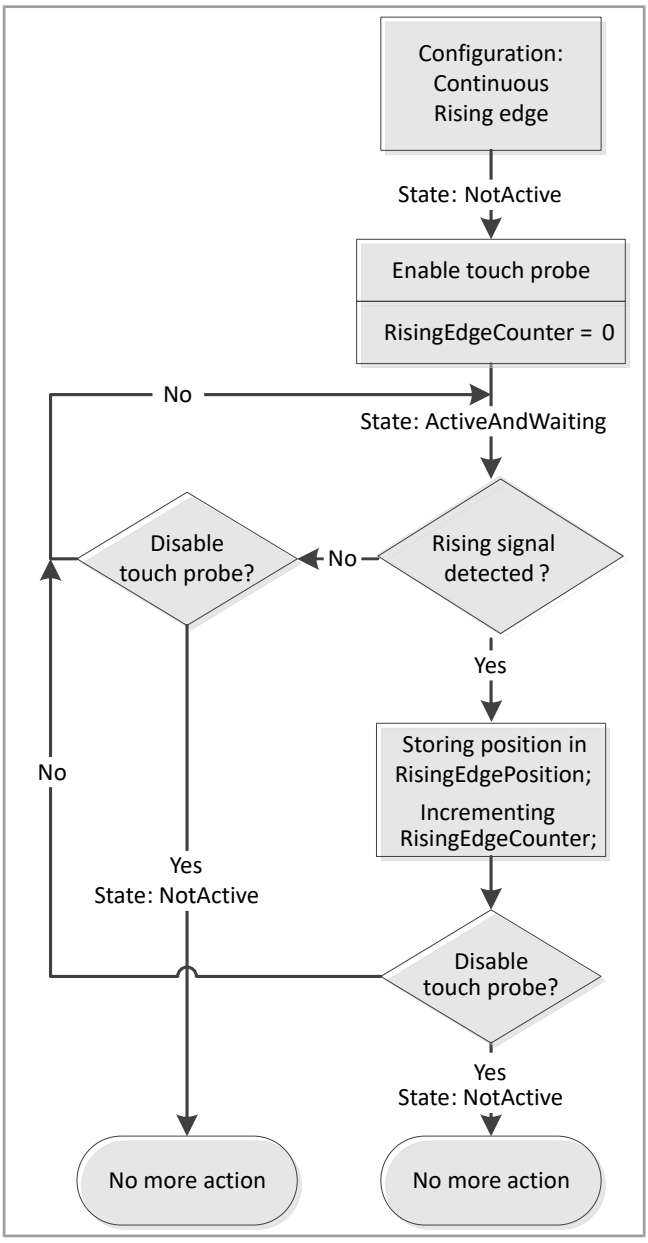


Fig. 8: *Flowchart for Touch Probe Configuration: Continuous Mode, Rising Edge*

Touch Probe Process

Here is how to use this feature:

1. Configure the touch probe application by setting the following parameters:
   – Mode (`single`, `continuous`)
   – Edge detection (`rising`, `falling`, `both`)
   – Position window limits (`WindowMin`, `WindowMax`)
2. Enable the touch probe function using the `Enable`command.

   ⇨ The*edge counter*is automatically reset to zero.
3. Check the Touch Probe status in **Single Mode** .

   ⇨ Initially, the status is " *active and wait* ".

   ⇨ The status changes to …*edge found* when:
   – The current actual position of the MCX axis has been saved in the *EdgePosition* parameter.– The counter for the configured edge (*EdgeCounter*) has been incremented.

4. Restart the touch probe function using an `enable` command. Wait at least two MCX cycles (typically 4 ms) before calling the Touch Probe `Enable` command again.

   OR

5. Use a `"Disable"` command to permanently exit the touch probe function.
6. Check the Touch Probe status in **Continuous Mode** .
   ⇨ Initially, the status is " *active and wait* ".
   ⇨ A change in the edge counter for the configured edge (…*EdgeCounter*) occurs when:
   – The current actual position of the MCX axis has been saved in the "..EdgePosition" parameter.
7. End the touch probe function using the `Disable` command.

⇨ The Touch Probe procedure is complete.

42.1 Syntax for MCAxis.TouchProbe

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis name>.TouchProbe[n].<Function>(optional <Parameter>);
```

42.2 Features of MCAxis.TouchProbe

This function offers the following functions:

– Configure()
– Enable()
– Disable()
– ReadConfiguration()
– ReadState()

42.2.1 Configure()

This function allows you to provide the **TouchProbe** function with the information that is appropriate for your application.

transfer parameter

Parameters	data type	Meaning
Edge	MCAxisTouchProbeEdges	The signal edge to be detected.
	MCAxisTouchProbeEdges.Rising	Rising flank
	MCAxisTouchProbeEdges.Falling	Falling edge
	MCAxisTouchProbeEdges.Both	Both flanks
Mode	Type MCAxisTouchProbeModes	*Touch Probe Mode*, which is to be used.
	MCAxisTouchProbeModes.Trigger-FirstEvent	*Single Mode*: one-time capture only.
	MCAxisTouchProbeModes.Continuous	*Continuous Mode*: continuous evaluation of the edge
WindowMin	Double	Lower limit of the position window
WindowMax	Double	Upper limit of the position window

Syntax

```
AX1.TouchProbe[1].Configure(MCAxisTouchProbeEdges.Rising, MCAxisTouchProbe-
Modes.TriggerFirstEvent,150.3,256.8);
AX1.TouchProbe[1].Configure(tpNewConfig.Edge,
                            tpNewConfig.Mode,
                            tpNewConfig.WindowMin,
                            tpNewConfig.WindowMax);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

42.2.2 Enable()

Use this function to enable the touch probe function.

This causes the MCX core to reset the touch-probe counters (`RisingEdgeCounter`, `FallingEdge-Counter`) to zero, which in turn sets the touch-probe status to *ActiveAndWaiting*.

In *Single Mode,* you can use the `Enable()` function to reactivate the touch probe function even without first issuing a `Disable()` command.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.TouchProbe[1].Enable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

42.2.3 Disable()

Use this function to exit the touch probe function.

transfer parameter

This function has no transfer parameters.

Syntax

```
AX1.TouchProbe[1].Disable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

42.2.4 ReadConfiguration()

This function retrieves the currently active touch probe configuration. The function passes data from the `MCAxisTouchProbeConfiguration` structure. The individual elements of the structure can also be read.

transfer parameter

Parameters	data type	Meaning
Edge	MCAxisTouchProbeEdges	The signal edge to be detected.
	MCAxisTouchProbeEdges.Rising	Rising flank
	MCAxisTouchProbeEdges.Falling	Falling edge
	MCAxisTouchProbeEdges.Both	Both flanks

Parameters	data type	Meaning
Mode	Type MCAxisTouchProbe-Modes	The touch probe mode to be used.
	MCAxisTouchProbeModes.Trig-gerFirstEvent	Single Mode: one-time capture only.
	MCAxisTouchProbe-Modes.Continuous	Continuous Mode: continuous evaluation of the edge
WindowMin	Double	Lower limit of the position window
WindowMax	Double	Upper boundary of the position window

Syntax

```
TPAktConfiguration := AX1.TouchProbe[1].ReadConfiguration();

TPConfiguredEdge := AX1.TouchProbe[1].ReadConfiguration().Edge;
```

state administration

The following table applies when reading the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

42.2.5 ReadState()

This function allows you to read the currently active Touch Probe status. The function returns data from a structure of type `MCAxisTouchProbeState`. The individual elements of the structure can also be read.

transfer parameter

Parameters	data type	Meaning
Trigger State	MCAxisTouchProbeTrigger-States	Status during the touch probe function.
	MCAxisTouchProbeTrigger-States.NotActive	The touch probe function is not active.
	MCAxisTouchProbeTrigger-States.ActiveAndWaiting	The touch probe function is active. No signal edge has been detected in the position window yet.
	MCAxisTouchProbeTrigger-States.RisingEdgeFound	The touch probe function is configured for a rising edge, and a rising edge has been detected. As a result, the current position of the MCX axis was copied to `RisingEdgePosition`, and `RisingEdgeCounter` was incremented
	MCAxisTouchProbeTrigger-States.FallingEdgeFound	The touch probe function is configured for a falling edge, and a falling edge has been detected. As a result, the current position of the MCX axis was copied to `FallingEdgePosition` and `FallingEdgeCounter` was incremented.
	MCAxisTouchProbeTrigger-States.BothEdgesFound	The touch probe function is configured for both edges, and both edges were detected one after the other. When the rising edge signal was detected, the current position of the MCX axis was copied to `RisingeEdgePosition` and `RisingEdgeCounter` was incremented. When the rising edge signal was detected, the current position of the MCX axis was copied to `FallingEdgePosition` and `FallingEdgeCounter` was incremented.
	MCAxisTouchProbeTrigger-States.NoEventInWindow	The current position has entered the position window once and has left it again. However, the touch probe function did not detect the configured signal edge.
RisingEdge-Counter	Int	Number of rising edges detected in the sensor signal. In single-mode, only the values 0 and 1 occur.
RisingEdge-Position	Double	Copied actual position at the rising edge of the sensor signal. After the function is activated, the value remains undefined until a rising edge is detected (specified in the user position unit).
Falling-Edge Counter	Int	Number of falling edges detected in the sensor signal. In single-mode, only the values 0 and 1 occur.
Falling Edge Position	Double	Copied actual position at the falling edge of the sensor signal. After the function is activated, the value remains undefined until a falling edge is detected (specified in the user position unit).

Syntax

```
TPAktStatus := AX1.TouchProbe[1].ReadState();

If AX1.TouchProbe[1].ReadState().TriggerState ==
MCAxisTouchProbeTriggerStates.RisingEdgeFound then
```

state administration

The following table applies when reading the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

42.3 Example task: MCAxis.TouchProbe

Here is an example task for a single axis that illustrates the process shown in the second flowchart above:

```
Var
    Run             : bool; //starts and stops the entire process (set by
the user)
    Trigger Position: Double;
    SignalState     : Enum TypeSignalState (undefined = 0, ok, notFound);
End_Var;

Task TouchProbe Autorun

AX1.Diagnostics.ClearErrors();
AX1.MoveHome.SetReference(5.0);

//AX1: axis modulo: range modulo = 0°.. 360°
//Touch Probe Configuration: Position Window 100° .. 180°
AX1.TouchProbe[1].Configure(MCAxisTouchProbeEdges.Falling,
                            MCAxisTouchProbeModes.TriggerFirstEvent,
                            100.0,
                            180.0);
Loop

When Run Continue;
    AX1.Power.Enable();
    When AX1.State.IsEnabled , continue;
    // Start the axis in endless mode
    AX1.MoveVelocity.Start(Directions.Positive,50.0,10000.0,10000.0);
    When AX1.State.IsRunning , Continue;
```

```
    Repeat
        //Enable touch probe function
        SignalState := TypeSignalState.undefined;
        TriggerdPosition := 0.0;
        AX1.TouchProbe[1].Enable();
        When AX1.TouchProbe[1].ReadState().TriggerState ==
MCAxisTouchProbeTriggerStates.ActiveAndWaiting Continue;

        When AX1.TouchProbe[1].ReadState().TriggerState ==
MCAxisTouchProbeTriggerStates.FallingEdgeFound Then
            TriggerdPosition :=
AX1.TouchProbe[1].ReadState().FallingEdgePosition;
            SignalState := TypeSignalState.ok
        Else_When AX1.TouchProbe[1].ReadState().TriggerState ==
MCAxisTouchProbeTriggerStates.NoEventInWindow Then
            SignalState := TypeSignalState.notFound;
        End_When;
        //Wait for modulo overflow
        When Ax1.Position.Actual > 350.0, Continue;
        When Ax1.Position.Actual < 5.0 Continue;

    Until Not Run;

    AX1.TouchProbe[1].Disable();
    AX1.MoveHalt.Start(MCAxisHaltModes.Normal,10000.0);
    When AX1.State.IsEnabled , continue;
    AX1.Power.Disable();
    When AX1.State.IsDisabled , continue;

End_Loop;
End_Task;
```

43 MCCouplingGroup

The **MCCouplingGroup** function allows you to define and use the position of one axis as a subordinate position processing unit for another axis. Possible applications for this include:

1. Move mechanically coupled axes independently

   There are mechanical arrangements in which two axes are coupled in such a way that the second axis rotates along with the first when the first is moved. To prevent the position from changing when the second axis is used, the position of the first axis can be locked to that of the second. In this way, the two axes are “decoupled” from the user’s perspective and can be moved independently of one another.

2. (Position) correction of an axis during movement

   By coupling a virtual axis with a physical axis, you can correct the reference position of the physical axis during movement based on external information (such as a print mark). Since the movement of the subordinate virtual axis is limited in terms of speed, acceleration, and jerk (this is a MovePtP motion), you can specify the behavior of the compensating motion.

You can also specify a conversion factor and a divisor for the coupling. A negative factor can be used to adjust the direction of the coupling's effect.

The coupling takes the transmission settings of the axes into account.

ⓘ Info

Requirements for creating a coupling group

The master axis can be coupled to only one slave axis. The functions include parameters that will make it possible to link multiple slave axes to a master axis in future updates.

Create a coupling group

To use a coupling group, you must first define it as a variable in the STX program:

```
Var
DMKorr: MCCouplingGroup('PressureMCorrection', AX10, mcMgr);
End_Var;
```

This call creates a new coupling group (`MCCouplingGroup`) using the following parameters:

Parameters	data type	Meaning
Groupname	String	Name of your choice.
MasterAxis	Pointer to MCAxis	An axis that is used as a parent axis by "child axes".
mcMgr	IMCAxisManager	An interface parameter that must be passed in this manner.

43.1 Syntax for MCCouplingGroup

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Variable of type MCCouplingGroup>.<Function or Access to
Value>(optional <Parameter1, Parameter 2, Parameter n>);
```

43.2 Features of MCCouplingGroup

This function offers the following functions:

– AddFollowerAxis()
– Redefine()
– Undefine()

43.2.1 AddFollowerAxis()

Use this function to add a secondary axis to the coupling group. This axis is subordinate to the reference axis specified in the variable definition. You also specify the translation for the sub-layers.

The coupling takes effect only after the Redefine() [▶ 242] function has been called.

transfer parameter

Parameters	data type	Meaning
Idx	Integer	Index of the secondary axis. Must always be `0` .
AxFollower	Pointer to MCAxis	Object of the follow axis. The leading axis is mounted below this axis.
Factor	Integer	Ratio for the transmission between the drive shaft and the driven shaft.
Divisor	Integer	Divisor for the transmission between the drive shaft and the driven shaft.

Syntax

```
DMKorr.AddFollowerAxis(0,AX1,1,5);
```

state administration

The function can be called from any state of the master and slave axes.

43.2.2 Redefine()

This function activates the coupling created using AddFollowerAxis() [▶ 242] .

transfer parameter

The function has no transfer parameters.

Syntax

```
DMKorr.Redefine();
```

state administration

The following table applies when calling the function for the master axis :

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

The function can be called in any state of the slave axis.

43.2.3 Undefine()

This function dissolves the coupling group. The axes are decoupled from one another. This deletes all internal position offsets.

transfer parameter

The function has no transfer parameters.

Syntax

```
DMKorr.Undefine();
```

state administration

The following table applies to the reference and follow axes when calling the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

43.3 Properties of the MCCouplingGroup

This function offers the following properties:

– FollowerCount
– MasterAxis
– FollowerAxis
– Name

43.3.1 FollowerCount

Description

data type	Meaning
Integer	Reads the number of defined secondary axis objects in the coupling group. This property anticipates future enhancements, as currently only a single follow-up axis can be added to a coupling group. Querying the property before calling the `AddFollower()` function returns the value 0. Querying the property after calling the `AddFollower()` function returns the value 1.

Syntax

```
NumberOfSub-Axes := DMKorr.FollowerCount;
```

state administration

The property can be read in any state of the master and slave axes.

43.3.2 MasterAxis

Description

data type	Meaning
Pointer to MCAxis	Reads the axis object that was defined as the reference axis in the coupling group.

Syntax

```
LeadingAxisObject := DMKorr.MasterAxis;
```

state administration

The property can be read in any state of the master and slave axes.

43.3.3 FollowerAxis[]

Description

NOTICE



STX exception error due to missing coupling

If you query this property without a child object being present, an exception error is generated in the STX program.

► Before retrieving the property, check whether a coupling exists.

data type	Meaning
Pointer to MCAxis	Reads the axis object that was defined as the follow-up axis in the coupling group. The array index is reserved for future extensions and must always be 0 for now.

Syntax

```
if DMKorr.FollwerAxis[0] != null then
Name_CouplingSequenceAxis := DMKorr.FollowerAxis[0].Info.Name;
end_if
```

state administration

The property can be read in any state of the master and slave axes.

43.3.4 Name

Description

data type	Meaning
String	Reads the name assigned to the coupling group during variable definition.

Syntax

```
LinkName := DMKorr.Name;
```

state administration

The property can be read in any state of the master and slave axes.

44 MCGantryGroup

The **MCGantryGroup** function (1:1 mechanically coupled axes) allows you to link two or more mechanically coupled drives that move a load together so that they function as a single axis. All axes must have the same mechanical configuration. The following guidelines apply:

– If a slave axis is assigned to a gantry group, it always receives the same position setpoint as the master axis and thus travels the same distance.
– The `Enable()` and `Disable()` functions are always executed on all axes of the gantry group.
  Each time the gantry group or its master axis `is enabled`, the actual and target positions of the gantry group's slave axes are set to the actual position of the master axis.
– The deviation from the actual position is monitored.
– It is possible to reference all axes involved together.
  This is advantageous when a single axis is not capable of moving the load on its own.
  Referencing is performed using the reference axis. The secondary axes are loaded at the reference point with the same reference value that the master axis determined.
– If the master axis is part of a technology group, the slave axes are treated the same as the master axis. They do not need to be specifically included in the technology group.

You can control the gantry group either by using the gantry group's object name followed by the label `"MasterAxis"` or by using the axis object of the master axis. Once the secondary axis has undergone coupling, it can no longer be operated directly. Operations on the secondary axis will result in an error message as long as the coupling is active.

ⓘ Info

Requirements for creating a coupling group

The master axis can be coupled to only one slave axis. The functions include parameters that will make it possible to link multiple slave axes to a master axis in future updates.

Create a portal group

To use a portal group, you must first define it as a variable in the STX program:

```
Var
Portal 1: MCGantryGroup('Portal1', AX1, mcMgr);
End_Var;
```

This call creates a new portal group (`MCGantryGroup`) using the following parameters:

Parameters	data type	Meaning
Groupname	String	Name of your choice.
MasterAxis	Pointer to MCAxis	The main axis in the portal group through which all commands are executed.
mcMgr	IMCAxisManager	An interface parameter that must be passed in this manner.

44.1 Syntax for MCGantryGroup

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Variable of type MCGantryGroup>.<Function or Access to
Value>(optional <Parameter 1, Parameter 2, Parameter n>);
```

44.2 Features of MCGantryGroup

This function offers the following functions:

– AddFollowerAxis()
– Redefine()
– Undefine()

44.2.1 AddFollowerAxis()

Use this function to add a follower axis to the portal group. This assigns the same setpoints to the axis as the master axis, and it can be controlled via the master axis.

The coupling takes effect only after the Redefine() [▶ 247] function has been called.

transfer parameter

Parameters	data type	Meaning
Idx	Integer	Index of the secondary axis.
AxFollower	Pointer to MCAxis	Object of the follow axis.

Syntax

```
DMKorr.AddFollowerAxis(0, AX2);
```

state administration

The function can be called from any state of the master and slave axes.

44.2.2 Redefine()

This function activates the coupling created using AddFollowerAxis() [▶ 247] .

transfer parameter

The function has no transfer parameters.

Syntax

```
Portal1.Redefine();
```

state administration

The following table applies to the **reference and secondary axes** when calling the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

44.2.3 Undefine()

This function dissolves the portal group. The axes are decoupled from one another.

transfer parameter

The function has no transfer parameters.

Syntax

```
Portal1.Undefine();
```

state administration

The following table applies to the reference and secondary axes when calling the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

44.3 Properties of MCGantryGroup

This function offers the following properties:

– GantryWindow[]
– ConfigGantryWindow[]
– Follower Count
– MasterAxis
– FollowerAxis[]
– Name

44.3.1 GantryWindow[i]

Description

data type	Meaning
Double	Reads or writes the maximum allowed positional deviation of the secondary axis with index `i` from the master axis in the portal group; specified in the user position unit. Default value: `10.0` If the permissible positional deviation is exceeded, the portal group enters an error state. The error response is configured using the `ConfigGantryWindow` property.

Syntax

```
Portal1.GantryWindow[0] := 20.0;
GantryWindow := Portal1.GantryWindow[0];
```

state administration

The property can be read and written in any state of the sequence axis.

44.3.2 ConfigGantryWindow[i]

Description

data type	Meaning
Integer	Reads or writes the error response of the portal group when the position deviation of the slave axis with index `i` exceeds the limit value `GantryWindow[i]` ; Default value: `5`
1	Triggers only one notification. Does not cause the axes to react.
4	The axes are stopped using the programmed ramp and remain in control mode after coming to a stop.
5	The axes are stopped using the emergency stop ramp and locked after coming to a stop (default)
6	The axes are locked immediately.

Syntax

```
Portal1.ConfigGantryWindow[0] := 5;
GantryConfig := Portal1.ConfigGantryWindow[0];
```

state administration

The property can be read and written in any state of the sequence axis.

44.3.3 FollowerCount

Description

data type	Meaning
Integer	Reads the number of defined secondary axis objects in the portal group.

Syntax

```
NumberOfFollowerAxes := Portal1.FollowerCount;
```

state administration

The property can be read in any state of the master and slave axes.

44.3.4 MasterAxis

Description

data type	Meaning
Pointer to MCAxis	Reads the axis object that has been defined as the reference axis in the portal group.

Syntax

```
ReferenceAxisObject := Portal1.MasterAxis;
```

state administration

The property can be read in any state of the master and slave axes.

44.3.5 FollowerAxis[i]

Description

NOTICE



STX exception error due to missing coupling

If you query this property without a child object being present, an exception is thrown in the STX program.

► Before retrieving the property, check whether a coupling exists.

data type	Meaning
Pointer to MCAxis	Reads the axis object defined in the portal group as a child axis with the index `i` .

Syntax

```
if Portal.FollwerAxis[0] != null then
Name_GantryFollowerAxis := Portal1.FollowerAxis[0].Info.Name;
end_if
```

state administration

The property can be read in any state of the master and slave axes.

44.3.6 Name

Description

data type	Meaning
String	Reads the name assigned to the portal group during variable definition.

Syntax

```
Gantryname := Portal1.Name;
```

state administration

The property can be read in any state of the master and slave axes.

44.4 Example Task: MCGantryGroup

Here is an example task for a portal group that includes various Motion API functions:

```
Var
//Definition of a gantry group with a name (for the group), a master axis
(typically one of the two axes), and an MC manager function
    Portal 1: MCGantryGroup('Portal1', AX1, mcMgr);
//Additional variables for the sample program
    Run: bool;
    Gantry_Number: int;
    GantryFenster: double;
    GantryConfig: int;
End_Var;

Task tGantry Autorun

//Ungrouping a portal group—this is not usually necessary, but it can be
useful, for example, if you want to move the axes individually.
    if Portal1.FollowerCount > 0 then
        Portal1.Undefine();
    end_if;

    //First, a follow-up axis is added to the group. You can also add
multiple secondary axes
    if Portal1.FollowerCount = 0 then
        Portal1.AddFollowerAxis(0, AX2);
        Portal1.Redefine();
    end_if;

    Gantry_Count := Portal1.FollowerCount; //FollowerCount indicates how
many follower axes belong to the portal group.

    Portal1.GantryWindow[0] := 20; //max. Tracking error between the master
and slave axes in user units
    GantryFenster := Portal1.GantryWindow[0]; //The value can also be read
back.
    //The allowed tracking error between the two axes is defined in
"GantryWindow[x]".
    //An array index must be specified because more than two axes can be
included in the gantry group.
    //If the programmed tolerance is exceeded, the gantry group enters an
error state
    //The error response can be configured:
    Portal1.ConfigGantryWindow[0] := 5; // 5 is the default value
    // The configuration can also be read back
    GantryConfig := Portal1.ConfigGantryWindow[0];

    //Referencing
    //Referencing is performed using the reference axis. If the reference is
found, the position of the secondary axis is also set to this value.
    //Each time "Enable" is called, the actual and target positions of the
slave axis are set to those of the master axis.

    if Portal1.MasterAxis.State.IsEnabledis false then
```

```
        Portal1.MasterAxis.Power.Enable();
        If Portal1.MasterAxis.State.IsEnabled , continue;
    end_if;
    Portal1.MasterAxis.MoveHome.Start(Directions.Positive,
ReferencingSwitchConfigs.RefSwitch,
                        ReferencingModes.SwitchingEdge_1Phase,
                        100000.0, 300.0, 10000.0, 100.0, 36000.0, 0.0, 0.0);
    When Portal1.MasterAxis.MoveHome.IsReferenceSet Continue;
    // AX1 can also be used instead of Portal1.MasterAxis

    Portal1.MasterAxis.Power.Disable();
    If Portal1.MasterAxis.State.IsDisabled , continue

Loop
    //Operation is controlled solely via the main axis.
    //The setpoints of the master axis are applied to the slave axis(es)
(including an internal offset that is determined during enablement).
When Run Continue;
    // This is how the gantry is used as an axis

        // Each time "Enable" is called, the actual and target positions of
the slave axis are set to those of the master axis.
        if Portal1.MasterAxis.State.IsEnabledis false then
            Portal1.MasterAxis.Power.Enable();
            If Portal1.MasterAxis.State.IsEnabled , Continue;
        end_if;

        //These commands move both axes back and forth between 1800.0 and
0.0
        Portal1.MasterAxis.MovePtp.Start(AxisPositioningModes.AbsNormal,1800
.0,1000.0,10000.0,10000.0,0.1);
        when Portal1.MasterAxis.IsInTargetWindow Continue;
            Delay(t#500ms);

        AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,0.0,1000.0,10000.0,
10000.0,0.1);
        when AX1.IsInTargetWindow Continue;
        Delay(t#500ms);
        if not run , then
            Portal1.MasterAxis.Power.Disable();
            If Portal1.MasterAxis.State.IsDisabled , Continue;
        end_if;
End_Loop;
End_Task;
```

45 MCGeo.Activate, MCGeo.Deactivate

You can use the **MCGeo.Activate** and **MCGeo.Deactivate** functions to activate and deactivate a path group. Changes to a path group are permitted in all operational states of the group, except when **it is inactive** .

Once a path group has been activated, all member routes are set to the **inactive** operational status. You can now only access these axes through the path group to which they are connected.

A path group can only be activated or deactivated if all member routes in that group are either in the **"enabled"** or **"disabled"** operational state, and all member routes have the same operational state. Otherwise, an error message will be displayed.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

45.1 Syntax for MCGeo.Activate, MCGeo.Deactivate

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Activate();
<Axis group name>.Deactivate();
```

45.2 Functions of MCGeo.Activate and MCGeo.Deactivate

This function offers the following functions:

– Activate()

– Deactivate()

45.2.1 Activate()

This function activates a path group provided that all member routes are in either the **"Blocked"** or **"Open"** state.

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.Activate();
```

state administration

The following table applies to the reference and follow axes when calling the function:

Initial state	Allowed	Final state
Inactive	√	Locked/Unlocked
Blocked	–	Blocked
Approved	–	Approved
Working	–	Working

Initial state	Allowed	Final state
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Check whether the new state has been reached in the program before issuing any further commands to the axes.

```
If geoGroup.State.IsInactiveis not true , continue;
```

Alternatively, you can also check the **IsEnabled** or **IsDisabled** status of the member axes—depending on the status of the group before the command was called.

45.2.2 Deactivate()

Use this feature to deactivate a path group. After this command, the group is in an **inactive**state. The member axes of the group can once again be controlled independently. They take over the operational status of the path group.

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.Deactivate();
```

state administration

The following table applies to the reference and follow axes when calling the function:

Initial state	Allowed	Final state
Inactive	–	Inactive
Blocked	√	Inactive
Approved	√	Inactive
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Check whether the new state has been reached in the program before issuing any further commands to the axes.

```
If geoGroup.State.IsInactive , continue;
```

Alternatively, you can also check the **IsEnabled** or **IsDisabled** status of the member axes—depending on the status of the group before the command was executed.

46 MCGeo.Buffer

The **MCGeo.Buffer** function allows you to manage the path group's buffer.

Movement commands from the path group are always buffered and executed; that is, they are stored in a queue and processed one after another. This also allows for the interpolation of movements and the programming of entire contours consisting, for example, of multiple linear and circular movements.

The condition for transitioning from one path movement to the next (path movement transition condition) is passed to the buffer in the motion command via the `MCGeoPathBlockEnd` parameter.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

46.1 Syntax for MCGeo.Buffer

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Buffer.<Function>(optional <Parameter>);
```

46.2 Functions of MCGeo.Buffer

This function offers the following functions:

- Clear()
- Suspend()
- Resume()
- UserIndex.Enable()
- UserIndex.Disable()
- UserIndex.Reset()

46.2.1 Clear()

This function clears the input buffer, with the following effects:

- A motion programmed with interpolation is decelerated along the interpolation polynomial.
- A movement that has already begun will be carried through to the end.

The `Power.Disable()`, `Power.Quickstop()`, and `Stop()` commands also clear the pre-emptive buffer.

transfer parameter

This function has no transfer parameters.

Syntax

```
geoGroup.Buffer.Clear();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

46.2.2 Suspend()

This function pauses the processing of the buffer at the next block transition, with the following effects:

– The current movement is completed.

– The next operation described in the pre-fetch buffer will not be initiated.

– The group remains in the *buffered* operating state.

– A motion programmed with interpolation is decelerated along the interpolation polynomial.

Once the block during which the `Buffer.Suspend()` function was called has been completed, the path group is in the **Enabled (Paused)** state.

Calling the Resume() [▶ 258] function resumes processing of the buffer.

transfer parameter

This function has no transfer parameters.

Syntax

```
geoGroup.Buffer.Suspend();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	–	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

46.2.3 Resume()

This function resumes processing of the input buffer that was paused by Suspend() [▶ 257] .

transfer parameter

This function has no transfer parameters.

Syntax

```
geoGroup.Buffer.Resume();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	–	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

46.2.4 UserIndex.Enable()

This option enables the path section counter. The path section counter is incremented when a programmed path movement of the path group is initiated. If grinding occurs between two path movements, the path section counter is incremented after the grinding is complete.

It can be reset to zero using the UserIndex.Reset() [▶ 259] command.

transfer parameter

This function has no transfer parameters.

Syntax

```
geoGroup.Buffer.UserIndex.Enable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved

Initial state	Allowed	Final state
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

46.2.5 UserIndex.Disable()

This function disables the path section counter.

transfer parameter

This function has no transfer parameters.

Syntax

```
geoGroup.Buffer.UserIndex.Disable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

46.2.6 UserIndex.Reset()

This function resets the path section counter to zero. The reset also enables the path section counter.

transfer parameter

This function has no transfer parameters.

Syntax

```
geoGroup.Buffer.UserIndex.Reset();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

46.3 Properties of MCGeo.Buffer

This function offers the following properties:

– UserIndex.Actual

46.3.1 UserIndex.Actual

Description

data type	Meaning
Integer	Reads the number of path segments moved since the path segment counter was last reset.

Syntax

```
NumberOfSections := geoGroup.Buffer.UserIndex.Actual;
```

state administration

The property can be read in any state.

47 MCGeo.Coordinates

The **MCGeo.Coordinates** function allows you to retrieve the current values for the path group movement, sorted by coordinates.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

47.1 Syntax for MCGeo.Coordinates

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<AxisGroupName>.Coordinates.<Access to Value>;
```

47.2 Properties of MCGeo.Coordinates

This function offers the following properties:

– Actual
– Setpoint
– Target

47.2.1 Actual

Description

This property returns the current 6D coordinate points of the path group.

The coordinate values are passed in a structure containing 3 directional coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
Actual	Coordinates6D	Actual value of the TCP's position in 6D spatial coordinates.
Actual.X	Coordinates6D.X (Double)	Current value of the TCP position in the X direction.
Actual.Y	Coordinates6D.Y (Double)	Current value of the TCP's position in the Y direction.
Actual.Z	Coordinates6D.Z (Double)	Current value of the TCP's position in the Z direction.
Actual.U	Coordinates6D.U (Double)	Current value of the TCP orientation in the U direction.
Actual.V	Coordinates6D.V (Double)	Current value of the TCP orientation in the V direction.
Actual.W	Coordinates6D.W (Double)	Current value of the TCP's orientation in the W-direction.

Syntax

```
Ist6D := geoGroup.Coordinates.Actual;
X := geoGroup.Coordinates.Actual.X;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

47.2.2 Setpoint

Description

Use this property to read the current 6D coordinate points of the setpoint generator for the path group.

The coordinate values are passed in a structure containing 3 directional coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
Setpoint	Coordinates6D	Desired value for the TCP position in 6D spatial coordinates.
Setpoint.X	Coordinates6D.X (Double)	Setpoint for the TCP position in the X direction.
Setpoint.Y	Coordinates6D.Y (Double)	Target value for the TCP's position in the Y direction.
Setpoint.Z	Coordinates6D.Z (Double)	Setpoint for the TCP's position in the Z direction.
Setpoint.U	Coordinates6D.U (Double)	Setpoint for the TCP orientation in the U direction.
Setpoint.V	Coordinates6D.V (Double)	Setpoint for the TCP's orientation in the V direction.
Setpoint.W	Coordinates6D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
Setpoint6D := geoGroup.Coordinates.Setpoint;
SetpointX := geoGroup.Coordinates.Setpoint.X;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

47.2.3 Target

Description

Use this property to read the current 6D coordinate points of the target for the current path movement of the path group.

The coordinate values are passed in a structure containing 3 directional coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
Target	Coordinates6D	The position of the TCP in 6D spatial coordinates.
Target.X	Coordinates6D.X (Double)	The X-coordinate of the TCP's position.
Target.Y	Coordinates6D.Y (Double)	The Y-coordinate of the TCP's position.
Target.Z	Coordinates6D.Z (Double)	The position of the TCP in the Z direction.
Target.U	Coordinates6D.U (Double)	The goal is to orient the TCP in the U direction.
Target.V	Coordinates6D.V (Double)	The goal is to align the TCP in the V direction.
Target.W	Coordinates6D.W (Double)	The goal is to orient the TCP in the W direction.

Syntax

```
TargetValue6D := geoGroup.Coordinates.Target;
TargetValueX := geoGroup.Coordinates.Target.X;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

48 MCGeo.Coupling

The **MCGeo.Coupling** function allows you to control the movement of a path group's tool center point based on the positions of up to six other independent MCX axes. In this process, the path group is coupled to the other axis(es) and follows their movement. For example, the gripper arm of a multi-axis robot can be synchro-nized with the movement of a conveyor belt from which it is supposed to pick up an object.

Types of couplings

There are two ways of motion control:

1. The path movement follows the MCX axes in a linear motion.
2. The path movement follows the MCX axes in a circular motion.

You can connect the path group to up to six MCX axes. Depending on the type of coupling selected, these axes control the movement of the path group in different ways.

Motion control for coupling type 1: linear path movement:

– The movement of the path group in the X direction follows the first axis specified in the coupling com-mand in a linear mode.
– The movement of the path group in the Y-direction follows the second axis specified in the coupling com-mand in a linear mode.
– The movement of the path group in the Z direction follows the third axis specified in the coupling com-mand in a linear mode.
– The U-orientation of the path group follows the fourth axis specified in the coupling command in a linear mode.
– The V-orientation of the path group follows the fifth axis specified in the coupling command in a linear mode.
– The W-orientation of the path group follows the sixth axis specified in the coupling command in a linear mode.

Motion control for coupling type 2: circular path movement:

This type of constraint also requires the center coordinates of the circle and the coordinate plane in which the circle is to lie.

The first 3 coupled axes define the circular motion in polar coordinates.

– The radius of the circular motion of the path group follows the first axis specified in the coupling com-mand.
– The angle of the circular motion of the path group follows the second axis specified in the coupling com-mand.
– The helix of the orbital motion of the path group follows the third axis specified in the coupling command in a linear mode.
– The U-orientation of the path group follows the fourth axis specified in the coupling command in a linear mode.
– The V-orientation of the path group follows the fifth axis specified in the coupling command in a linear mode.
– The W-orientation of the path group follows the sixth axis specified in the coupling command in a linear mode.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

48.1 Syntax for MCGeo.Coupling

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Coupling.<Function or Access to Value>(optional <Parameter 1, Parameter 2, Parameter n>);
```

48.2 Features of MCGeo.Coupling

This function offers the following functions:

- Couple()
- Decouple()
- GetPositioningMode()
- GetCoupleType()
- GetCoupleMode()
- GetDecoupleMode()

48.2.1 Couple()

Use this function to configure the coupling between the MCX axes and the path group and establish the coupling. This process requires that the path group be in the **"Enabled"** operational state. When coupled, the path group's operating state is always **"Running"**, and it can no longer be moved using the `Move` commands.

transfer parameter

Parameters	data type	Meaning
Positioning Mode	MCGeoPositioning-Modes (optional)	Defines how the group tracks the positions of the MCX axes.
	MCGeoPositioning-Modes.Absolute	The motion sequence follows the positions of the MCX axes based on the same values for the target position and the offsets: *Target position of the TCP coordinate = target position of the MCX axis + offset*. When coupling begins, the position values of the TCP and the MCX axes involved (shifted by the offsets) must match to prevent jerky movement of the TCP.
	MCGeoPositioning-Modes.Relative	The array contains the positions of the MCX axes, along with their starting positions and the offsets of the MCX axes: *Target position of the TCP coordinate = Target position of the MCX axis + Starting position of the TCP coordinate + Offset*

Parameters	data type	Meaning
CoupleType	MCGeoCoupleTypes (optional)	Type of TCP movement relative to the MCX axes.
	MCGeoCouple-Types.Linear	The target positions of the MCX axes serve directly as the reference values for the target positions of the path coordinates.
	MCGeoCouple-Types.CircularXY	The target positions of the MCX axes serve as reference points for the polar coordinates (see above) of a circular motion in the XY plane.
	MCGeoCouple-Types.CircularYZ	The target positions of the MCX axes serve as reference points for the polar coordinates (see above) of a circular motion in the YZ plane.
	MCGeoCoupleTypesCir-cularZX	The target positions of the MCX axes serve as reference points for the polar coordinates (see above) of a circular motion in the ZX plane.
Couple Mode	MCGeoCouple-Modes.Immediate (optional)	The coupling will be performed immediately.
CoupleID_1	Integer (optional)	Object ID of the first MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 109]).
CoupleID_2	Integer (optional)	Object ID of the second MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 109]).
CoupleID_3	Integer (optional)	Object ID of the third MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 109]).
CoupleID_4	Integer (optional)	Object ID of the fourth MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 109]).
CoupleID_5	Integer (optional)	Object ID of the fifth MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 109]).
CoupleID_6	Integer (optional)	Object ID of the sixth MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 109]).
Offset_1	Double (optional)	Offset between the target position of the first MCX axis and the target value for the corresponding path coordinate.
Offset_2	Double (optional)	Offset between the target position of the second MCX axis and the target value for the corresponding path coordinate.
Offset_3	Double (optional)	Offset between the target position of the third MCX axis and the target value for the corresponding path coordinate.
Offset_4	Double (optional)	Offset between the target position of the fourth MCX axis and the target value for the corresponding path coordinate.
Offset_5	Double (optional)	Offset between the target position of the fifth MCX axis and the target value for the corresponding path coordinate.
Offset_6	Double (optional)	Offset between the target position of the sixth MCX axis and the target value for the corresponding path coordinate.
CenterPointX	Double (optional)	X-coordinate of the circle center point when `CoupleType` is set to `CircularXY` or `CircularZX` .
CenterPointY	Double (optional)	The Y-coordinate of the circle center point when`CircularXY` or `CircularYZ` is selected as `the CoupleType` .

Parameters	data type	Meaning
CenterPointZ	Double (optional)	Z-coordinate of the circle center point if the CoupleType is set to `CircularYZ` or `CircularZX`.

Syntax

```
geoGroup.Coupling.Couple(MCGeoPositioningModes.Absolute, MCGeoCouple-
Types.Linear, MCGeoCoupleModes.Immediate, AX1.Info.Objec-
tID,,,,,,10.0,,,,,,);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Working
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

48.2.2 Decouple()

transfer parameter

Parameters	data type	Meaning
Mode	MCGeoDecoupleModes.Immediate (optional)	Decouples the path group from the independent MCX axes to which it was connected.

Syntax

```
geoGroup.Coupling.Decouple(MCGeoDecoupleModes.Immediate);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	–	Approved
Working	√	Approved
Interrupted	–	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

48.2.3 GetPositioningMode()

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the current positioning mode of the paired path group. The return value is of type `MCGeoPositioningModes`.

Syntax

```
CouplePM := geoGroup.Coupling.GetPositioningMode(AX1);
```

state administration

The function can be read in any state.

48.2.4 GetCoupleType()

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the current coupling type of the coupled path group. The return value is of type `MCGeoCoupleTypes`.

Syntax

```
CoupleT := geoGroup.Coupling.GetCoupleType(AX1);
```

state administration

The function can be read in any state.

48.2.5 GetCoupleMode()

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the current coupling mode of the coupled path group. The return value is of type `MCGeoCoupleModes`.

Syntax

```
CoupleM := geoGroup.Coupling.GetCoupleMode(AX1);
```

state administration

The function can be read in any state.

48.2.6 GetDecoupleMode()

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the current decoupling mode of the coupled path group. The return value is of type `MCGeoDecoupleModes`. **So far, this has always been** `"Immediate"` **(**`1`**).**

Syntax

```
DecoupleM := geoGroup.Coupling.GetDecoupleMode(AX1);
```

state administration

The function can be read in any state.

48.3 Properties of MCGeo.Coupling

This function offers the following properties:

– CoupleState

48.3.1 CoupleState

Description

property	data type	Meaning
CoupleState	MCGeoCoupleStates	Reads the couple status of the path group.
	MCGeoCoupleStates.Decoupled	The path group is not connected to other axes.
	MCGeoCoupleStates.Coupled	The path group is in a coupled state.

Syntax

```
CoupleStatus := geoGroup.Coupling.CoupleState;
```

state administration

The property can be read in any state.

49 MCGeo.Diagnostics

The **MCGeo.Diagnostics** function allows you to manage error handling for a path group.

The Motion Control eXtended system distinguishes between *errors* and *messages*. Errors trigger a system response, whereas messages do not. Errors and messages are logged in separate buffers, from which they can be read. The buffer collects the events that have occurred since the last call to the `ClearErrors`() function (see ClearErrors() [▶ 40]).

The meanings of the error and message numbers are listed in the JetSym online help under “MotionControl Errors.”

49.1 Syntax for MCGeo.Diagnostics

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Path group name>.Diagnostics.<Function or Access to Value>(optional
<Parameter>);
```

49.2 Features of MCGeo.Diagnostics

This function offers the following functions:

– ClearErrors()
– Software.ClearErrors()
– Software.GetErrorCode()
– Software.GetMessageCode()

49.2.1 ClearErrors(), Software.ClearErrors() – path group

If errors occur, the functions acknowledge anerror condition in the path group. The entries in the error buffer are deleted.

Messages do not trigger an error response and therefore do not need to be acknowledged. For messages, these two functions cause the entries in the message buffer to be deleted.

transfer parameter

These functions do not have any transfer parameters.

Syntax

```
geoGroup.Diagnostics.ClearErrors();
geoGroup.Diagnostics.Software.ClearErrors();
```

ⓘ Info

Scope of the order

These commands clear all errors assigned by the group. These can be errors triggered by a group object, as well as errors reported by the group axes.

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	–	Blocked

Initial state	Allowed	Final state	
Approved	–	Approved	
Working	–	Working	
Interrupted	–	Interrupted	
Error: Approved	√	Approved	The system will only switch to this state once the cause of the error has been resolved. Otherwise, the error condition may persist or recur immediately.
Error: blocked	√	Blocked	

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

49.2.2 Software.GetErrorCode()

This function returns the error number at the error buffer location specified by `BufferIndex` . If no `BufferIndex` is provided, the number of the most recently recorded error is returned (`BufferIndex` = `0`).

transfer parameter

Parameters	data type	Meaning
BufferIndex	Integer (optional)	The `BufferIndex` parameter specifies the position in the error buffer from which the error number is to be read. The maximum value of the `BufferIndex` parameter can be retrieved via the `ErrorCount` property.

Syntax

```
geoGroup.Diagnostics.Software.GetErrorCode();
```

state administration

The function can be read in any state.

49.2.3 Software.GetMessageCode()

This function returns the message number at the message buffer location specified by `BufferIndex` . If no `BufferIndex` is provided, the number of the most recently recorded message is returned (`BufferIndex` = `0`).

transfer parameter

Parameters	data type	Meaning
BufferIndex	Integer (optional)	The `BufferIndex` specifies the position in the message buffer from which the message number is to be read. The maximum value of the `BufferIndex` parameter can be retrieved via the `MessageCount` property.

Syntax

```
geoGroup.Diagnostics.Software.GetMessageCode();
```

state administration

The function can be read in any state.

49.3 Properties of MCGeo.Diagnostics

This function offers the following properties:

– Software.ErrorCount
– IsErrorPending
– Software.IsErrorPending
– Software.MessageCount
– IsMessagePending
– Software.IsMessagePending

49.3.1 Software.ErrorCount

Description

property	data type	Meaning
Software.ErrorCount	Int	Reads the number of errors logged since the last time the `ClearErrors()` or `Software.ClearErrors()` function was called, or since the controller was restarted.

Syntax

```
ErrorCount := geoGroup.Diagnostics.Software.ErrorCount;
```

state administration

The property can be read in any state.

49.3.2 IsErrorPending, Software.IsErrorPending

Description

data type	Meaning
Bool	Check whether there is an error in the MCX system. This is equivalent to comparing `ErrorCount` to `0`.

Syntax

```
If geoGroup.Diagnostics.IsErrorPending then
...
End_if;
```

state administration

The property can be queried in any state.

49.3.3 Software.MessageCount

Description

data type	Meaning
Integer	Reads the number of logged messages since the last time the `ClearErrors()` or `Software.ClearErrors()` function was executed, or since the controller was restarted.

Syntax

```
MessageCount := geoGroup.Diagnostics.Software.MessageCount;
```

state administration

The property can be queried in any state.

49.3.4 IsMessagePending, Software.IsMessagePending

Description

data type	Meaning
Bool	Check whether there is a message in the MCX system. This is equivalent to comparing `MessageCount` to `0`.

Syntax

```
If geoGroup.Diagnostics.IsMessagePending then
...
End_if;
```

state administration

The property can be queried in any state.

49.4 Example task for MCGeo.Diagnostics

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    GeoError: Array[10] of INT; //Variables for displaying the error buffer
    i: INT; //Loop variable
    AcknolegeError: Bool; // Auxiliary flag for an acknowledgment condition

End_Var;

Task Sample_Error Autorun

Loop

    //Clear error variables for display
    For i := 0 To 9 Do
        GeoError[i] := 0;
    End_For;
    AcknowledgeError := False;

    When geoGroup.Diagnostics.IsErrorPending , continue;

        Taskbreak Automatic;
        // Add additional error handling here

        //Store errors in variables, e.g., for display purposes
        For i := 0 To geoGroup.Diagnostics.Software.ErrorCount- 1 Do
            GeoError[i] := geoGroup.Diagnostics.Software.GetErrorCode(i);
        End_For;

    // Bugs have been fixed
    When AcknowledgeError , continue;
```

```
        geoGroup.Diagnostics.ClearErrors();
        Automatictask restart ;
End_Loop;

End_Task;
```

50 MCGeo.Events

The **MCGeo.Events** function allows you to manage the behavior of flags that are set based on the sequence of movements in the path group's pre-buffer.

ⓘ Info

The `SetFlag()` function in conjunction with the input buffer

The `event` commands are also included in the header buffer. A condition for setting a flag (using the `SetFlag()` function) must be written to the preamble buffer before the path segment to which it refers.

Working with the pre-buffer makes it possible to trace path contours that consist of multiple path segments and/or merge into one another through overlapping. The functions of the **Events** function allow you to have the MCX kernel set flags. The flags depend on which path segment is currently being moved in the pre-run buffer and where the movement is currently located within that path segment. You can use these flags to control actions in the STX program, such as opening a metering valve.

When the path group is activated, all flags are cleared.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.**These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

50.1 Syntax for MCGeo.Events

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Events.<Function or Access to Value>(optional <Pa-
rameter);
```

50.2 Features of MCGeo.Events

This function offers the following functions:

- SetFlag()
- ReadFlags()
- ReadFlag()
- ClearFlags()

50.2.1 SetFlag()

This function instructs the MCX kernel to set a flag in the pre-run buffer when executing the following path segment, based on the transfer parameter.

If the condition at the end of the motion is selected for a motion with interpolation, the flag is set at the beginning of the interpolation polynomial.

transfer parameter

Parameters	data type	Meaning
BlockCondition	MCGeoEventConditions	A condition that defines at which point in the motion the flag with the `EventNo` is set.
	MCGeoEventConditions.BlockStart	The flag is set when the movement begins.
	MCGeoEventConditions.BlockEnd_InPosState	The flag is set at the end of the movement, as soon as the setpoint generator has reached the target.
	MCGeoEventConditions.BlockEnd_InPosState_DestWnd	The flag is set before the end of the movement, as soon as the target window is reached.
	MCGeoEventConditions.BlockEnd_InPosState_DestWnd_Delay	The flag is set at the end of the movement and after the programmed wait time.
Event No.	Integer (optional)	Flag number (1 ... 16), which is to be set under the above condition.

Syntax

```
geoGroup.Events.SetFlag(MCGeoEventConditions.BlockStart,2);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	X	Blocked
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

50.2.2 ReadFlag()

This function is used to read the status of a specific event flag.

transfer parameter

Parameters	data type	Meaning
Event No.	Integer	Flag number (1 ... 16), whose status is to be read.

Syntax

```
If geoGroup.Events.ReadFlag(2) then;
```

state administration

The function can be called from any state.

50.2.3 ReadFlags()

This function reads the status of all 16 event flags. The function returns a value of the Integer data type. The last 16 bits of this value correspond to the flags numbered 1 through 16. (Bit 0 = EventFlag 1, ...).

transfer parameter

The function has no transfer parameters.

Syntax

```
FlagWord := geoGroup.Events.ReadFlags();
```

state administration

The function can be called from any state.

50.2.4 ClearFlags()

This function clears the path group flags passed in the event mask.

Clearing the flags is not cached in the preprocess buffer but is executed immediately.

transfer parameter

Parameters	data type	Meaning
EventMask	Integer	Bit 0 ... 15 correspond to flags 1 ... Assigned to 16. A set bit clears the corresponding event flag.

Syntax

```
geoGroup.Events.ClearFlags(0x0001);
```

state administration

The following table applies when writing the function:

Condition Monitoring

If a `Deactivate()` command immediately follows a `ClearFlags()` command, the corresponding flags may not be cleared because the `Deactivate()` command has higher priority when processed in the same MCX cycle. However, since the `ClearFlags()` function is not executed in the **inactive** state, it makes sense in this case to check whether the relevant flags have been cleared before issuing the `Deactivate()` command.

50.3 Example task: MCGeo.Events

Here is an example task for a single axis that includes various Motion API functions:

```
Task: Path Movement - Autorun
...
    //Start three linear movements - set EventFlag 2 at the start of the
second movement
    geoGroup.Power.Enable();
    When geoGroup.State.IsEnabled , continue;
    geoGroup.Mechanism.Limitations.MaxPathAcceleration := 12000;
    geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute,
MCGeoPathBlockEnd.InPosState,300,20,0,0.1,100,50);
```

```
geoGroup.Events.SetFlag(MCGeoEventConditions.BlockStart, 2);
    geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute,
MCGeoPathBlockEnd.InPosState,300,10,500,0.1,150,150);
    geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute,
MCGeoPathBlockEnd.InPosState,300,10,0,0.1,0,0);
...
End_Task;


Task: Set up autorun

    If geoGroup.Events.ReadFlag(2) Then
        oDosage := true; //Enable dosing at the start of the second section
of path
    End_if;
...
End_Task;
```

51 MCGeo.Info

The **MCGeo.Info** function allows you to identify the group objects in MotionControl.

This information was entered by the user in the Hardware Manager and motion setup within the project.

51.1 Syntax for MCGeo.Info

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Info.<Property>;
```

51.2 Properties of the function

This function offers the following properties:

– Name
– ObjectID
– ClassID

51.2.1 Name

Description

data type	Meaning
String	Retrieves the name of the path group as defined in the hardware configuration. This is helpful if you want to use the name in the trace or on an external display.

Syntax

```
GeoName := Geo.Info.Name;
```

In this case , `GeoName` contains the string "`Geo`".

state administration

The property can be read in any state.

51.2.2 ObjectID

Description

data type	Meaning
Integer	Retrieves the internal control number of the path group. This property is typically only required in applications that use the MC Manager's features. One possible use case is described in the corresponding section MCManager (MCMgr) [▶ 401] .

Syntax

```
Composite ID := geoGroup.Info.ObjectID;
```

state administration

The property can be read in any state.

51.2.3 ClassID

Description

data type	Meaning
MCClassIDs	Queries the class of an MC object. An MC object can be an axis, a technology group, or a path group. This property is typically only required in applications that use the MC Manager's features. One possible use case is described in the corresponding section MCManager (MCMgr) [▶ 401] .

Syntax

```
ClassID := geoGroup.Info.ClassID;
```

state administration

The property can be read in any state.

51.3 Example task: MCGeo.Info

Here is an example task for a single axis that includes various Motion API functions:

After the example task has been executed, the trace displays whether each path group is in the **"Running"** state or not. The groups are listed by their group names.

```
function TraceRunningState(mcGeoGroup: pointer to MCGeo)
    if mcGeoGroup.State.IsRunning then
        trace(mcGeoGroup.Info.Name + 'is running$n');
    else
        trace(mcGeoGroup.Info.Name + 'is not running$n');
    end_if;
end_function;

task Sample1 // example task
var
    nGeoIndex:    int;
end_var;
    for nGeoIndex := 0 to mcMgr.GeoCount - 1 , incrementing by 1 , do
        TraceRunningState(mcMgr.GeoList[nGeoIndex]);
    end_for;
end_task;
```

52 MCGeo.Mechanism.Kinematics

The **MCAxis.Mechanism.Kinematics** function allows you to read the mechanical design of a path group and, in exceptional cases, configure it. Configuration is typically performed in MotionSetup during commissioning.

The mechanical configuration of a path group must be known to the controller so that the units and kinematics can be calculated correctly in the MCX kernel. In JetSym this is done using the definition in the Hardware tree and the motion setup. This data is saved in the *MCGeo.ini* file for the group and loaded into the controller. During bootup, this information is read from the ini file and is available when the application program starts.

Special Case: Changing the Kinematics Configuration Using `Configure`

In rare exceptional cases, the kinematics configuration can be modified using the `Configure` subfunctions. Please note that in this case, the mechanical parameters defined in the JetSym project are not used in the machine. This can lead to confusion, for example, during a service call. Since these mechanical parameters in a machine do not normally change during operation, it does not make sense to adjust them while the application program is running.

52.1 Syntax for MCGeo.Mechanism.Kinematics

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Mechanism.Kinematics.<Function or Access to
Value>(optional <Parameter 1, Parameter 2, Parameter n>);
```

52.2 Properties of MCGeo.Mechanism.Kinematics

This function offers the following properties of the `Mechanism` function:

– Cartesian2D
– Cartesian3D
– Scara

52.2.1 Cartesian2D

Description

Use the `offset` and `InvertDirection` properties to read the currently valid kinematic parameters of the path group coordinates.

property	data type	Meaning
X.Offset	Double	Offset of the linear position in the X direction.
X.InvertDirection	Bool	Reversal of the counting direction for the linear position in the X-direction.
Y Offset	Double	Offset of the linear position in the Y direction.
Y.InvertDirection	Bool	Reversal of the counting direction for the linear position in the Y-direction.

Syntax

```
XOffset := geoGroup.Mechanism.Kinematics.Cartesian2D.X.offset;
```

state administration

The property can be read in any state.

52.2.2 Cartessian3D

Description

Use the `offset` and `InvertDirection` properties to read the currently valid kinematic parameters of the path group coordinates.

property	data type	Meaning
X.Offset	Double	Offset of the linear position in the X direction.
X.InvertDirection	Bool	Reversal of the counting direction for the linear position in the X-direction.
Y Offset	Double	Offset of the linear position in the Y direction.
Y.InvertDirection	Bool	Reversal of the counting direction for the linear position in the Y-direction.
Z-offset	Double	Displacement of the linear position in the Z-direction.
Z.InvertDirection	Bool	Reversal of the counting direction of the linear position in the Z-direction.

Syntax

```
XOffset := geoGroup.Mechanism.Kinematics.Cartesian3D.X.offset;
```

state administration

The property can be read in any state.

52.2.3 Scara

The following figure schematically illustrates the basic concepts of path kinematics using a SCARA robot as an example.

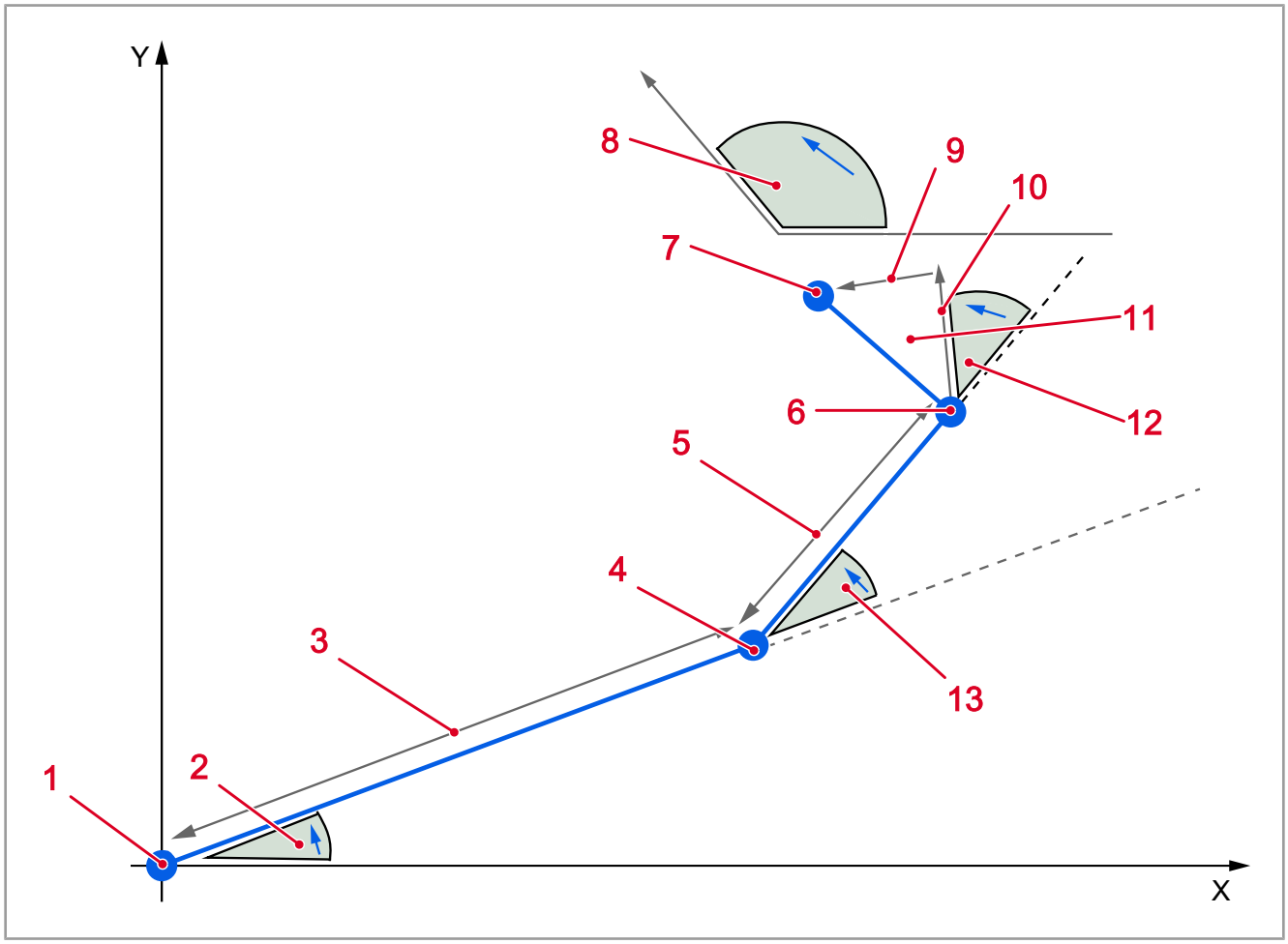


Fig. 9: SCARA kinematics

1	Shoulder
2	Shoulder angle (Motor 1)
3	Upper arm length
4	Elbow
5	Forearm length
6	Wrist
7	Tool Center Point, TCP = (X, Y, Z)
8	Orientation (U)
9	TCP_Offset.y
10	TCP_Offset.x
11	TCP angle = shoulder angle + elbow angle + wrist angle + TCP angle
12	Wrist angle (Motor 4)
13	Elbow angle (Motor 2)

Description

Use the `offset` and `InvertDirection` properties to read the currently valid kinematic parameters of the path group coordinates.

property	data type	Meaning
Shoulder offset	Double	Offset of the shoulder axis.
Shoulder.InvertDirection	Bool	Reversal of the direction of measurement for the shoulder axis.
Shoulder.ArmLength	Double	Arm length from the shoulder axis (upper arm).
Elbow offset	Double	Offset of the elbow axis.
Ellbow.InvertDirection	Bool	Reversal of the direction of measurement of the elbow axis.
Elbow to Arm's Length	Double	Arm length from the elbow axis (forearm).
Vertical Offset	Double	Offset of the vertical axis.
Vertical.InvertDirection	Bool	Reversal of the counting direction of the vertical axis.
Wrist offset	Double	Shift in the position of the hand axis.
Wrist.InvertDirection	Bool	Reversal of the counting direction of the hand axis position.
Wrist.TcpOffsetX	Double	Offset of the tool center point from the center of the hand axis in the X direction.
Wrist.TcpOffsetY	Double	Offset the tool center point from the center of the hand axis in the Y direction.

Syntax

```
HandOffset := geoGroup.Mechanism.Kinematics.Scara.Wrist.offset;
```

state administration

The property can be read in any state.

52.3 Configuration features of MCGeo.Mechanism.Kinematics

In the rare exceptional cases mentioned, you can configure the kinematic parameters from within the STX program. For this purpose, the "Configure" submenu is available in each case:

– Cartesian2D.X.Configure(), Cartesian2D.Y.Configure()
– Cartesian3D.X.Configure(), Cartesian3D.Y.Configure()
– Scara.Shoulder.Configure(), Scara.Elbow.Configure()
– Scara.Vertical.Configure()
– Scara.Wrist.Configure()

52.3.1 Cartesian2D.X.Configure(), Cartesian2D.Y.Configure()

The `Configure()` function passes the kinematic parameters for defining the kinematics of a coordinate axis in the path group.

If no parameter is passed, the last value passed is used.

transfer parameter

Parameters	data type	Meaning
offset	Double (optional)	Linear position offset.
InvertDirection	Modes (optional)	Reversal of the counting direction of the linear position.
	Modes.On	Enables reverse counting.
	Modes.Off	Disables the count direction reversal.

Syntax

```
geoGroup.Mechanism.Kinematics.Cartesian2D.X.Configure(1200.0, Modes.Off);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

52.3.2 Cartesian3D.X.Configure(), Cartesian3D.Y.Configure()

The `Configure()` function passes the kinematic parameters for defining the kinematics of a coordinate axis in the path group.

If no parameter is passed, the last value passed is used.

transfer parameter

Parameters	data type	Meaning
offset	Double (optional)	Linear position offset.
InvertDirection	Modes (optional)	Reversal of the counting direction of the linear position.
	Modes.On	Enables reverse counting.
	Modes.Off	Disables the reversal of the counting direction.

Syntax

```
geoGroup.Mechanism.Kinematics.Cartesian3D.X.Configure(1200.0, Modes.Off);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

52.3.3 Scara.Shoulder.Configure(), Scara.Elbow.Configure()

The `Configure()` function passes the kinematic parameters for defining the kinematics of a coordinate axis in the path group.

transfer parameter

Parameters	data type	Meaning
offset	Double (optional)	Linear position offset.
InvertDirection	Modes (optional)	Reversal of the counting direction of the linear position.
	Modes.On	Enables reverse counting.
	Modes.Off	Disables the count direction reversal.
ArmLength	Double (optional)	Arm length along the shoulder-to-elbow axis.

Syntax

```
geoGroup.Mechanism.Kinematics.Scare.Shoulder.Configure(10.0,
Modes.Off,205.5);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

52.3.4 Scara.Vertical.Configure()

The `Configure()` function passes the kinematic parameters for defining the kinematics of a coordinate axis in the path group.

transfer parameter

Parameters	data type	Meaning
offset	Double (optional)	Linear position offset.
InvertDirection	Modes (optional)	Reversal of the counting direction of the linear position.
	Modes.On	Enables reverse counting.
	Modes.Off	Disables the reversal of the counting direction.

Syntax

```
geoGroup.Mechanism.Kinematics.Scare.Vertical.Configure(25.0, Modes.Off);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

52.3.5 Scara.Wrist.Configure()

The `Configure()` function passes the kinematic parameters for defining the kinematics of a coordinate axis in the path group.

transfer parameter

Parameters	data type	Meaning
offset	Double (optional)	Linear position offset.
InvertDirection	Modes (optional)	Reversal of the counting direction of the linear position.
	Modes.On	Enables reverse counting.
	Modes.Off	Disables the count direction reversal.
TcpOffsetX	Double (optional)	Offset the tool center point from the center of the hand axis in the X direction.
TcpOffsetY	Double (optional)	Offset of the tool center point from the center of the hand axis in the Y direction.

Syntax

```
geoGroup.Mechanism.Kinematics.Scare.Wrist.Configure(10.0,
Modes.Off,50.0,50.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53 MCGeo.Mechanism.Limitations

The **MCGeo.Mechanism.Limitations** function allows you to pass the mechanical constraints of the machine's path group to the MCX controller. These are used as limit values for motion programming.

If the limits of the working area are exceeded during the programming of a motion, an error is generated and the motion is not initiated.

When calculating the motion profile, the maximum values for velocity, acceleration, and—in the case of jerk-limited motion types—jerk are taken into account. If the motion command results in values for speed, acceleration, and jerk that exceed the limits, these values are automatically capped at the maximum limits without an error message.

To illustrate the kinematic parameters, the following figure shows a schematic mechanical configuration using a SCARA robot as an example.

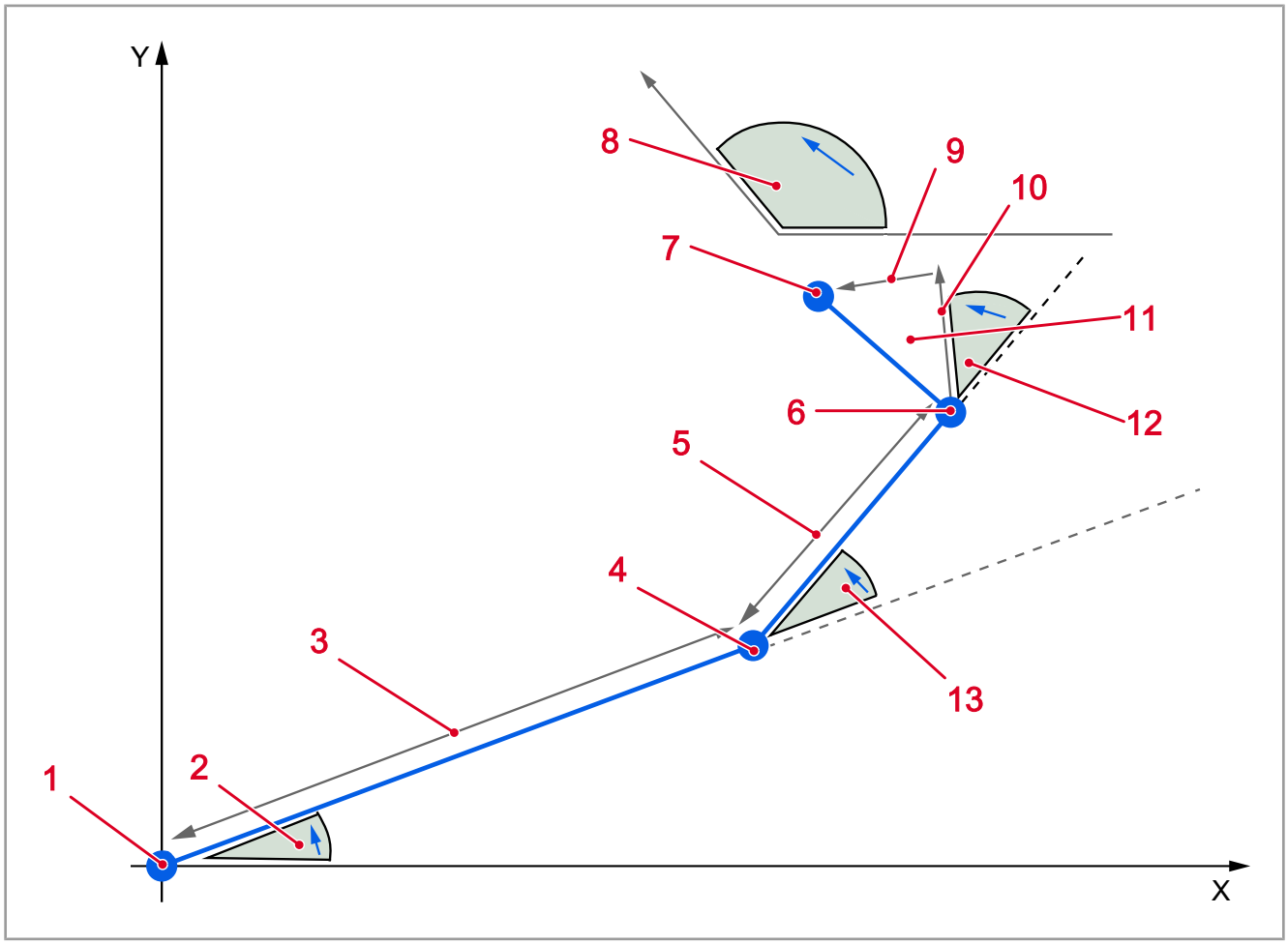


Fig. 10: SCARA kinematics

1	Shoulder
2	Shoulder angle (Motor 1)
3	Upper arm length
4	Elbow
5	Forearm length
6	Wrist
7	Tool Center Point, TCP = (X, Y, Z)
8	Orientation (U)
9	TCP_Offset.y

10	TCP_Offset.x
11	TCP angle = shoulder angle + elbow angle + wrist angle + TCP angle
12	Wrist angle (Motor 4)
13	Elbow angle (Motor 2)

The orientation of the Tool Center Point (TCP) is defined by the so-called **Cardan angles**, in the axis order ***z – y' – x''***, relative to the base coordinate system.

Calculation of the cardan angle

To derive the TCP coordinate system ***(x''', y''', z''')*** from the base coordinate system (x, y, z), the following procedure is used:

1. First, the fixed coordinate system is rotated by an angle U around the ***z***-axis. This results in the coordinate system ***(x', y', z')***.
2. This is now rotated by an angle V about the ***y***'-axis, resulting in the coordinate system ***(x'', y'', z'')*** .
3. Finally, this coordinate system is rotated by an angle W around the ***x***''-axis. This defines the TCP coordinate system ***(x''', y''', z''')***.

Meaning of the cardan angle

– The angle U (also known as the yaw angle) describes the orientation of the TCP relative to the ***x-y*** plane of the base coordinate system.
– The angle V (also known as the pitch angle) describes the orientation of the TCP relative to the ***y-z*** plane of the Cartesian coordinate system.
– The angle W (also known as the roll angle) describes the rotation of the TCP around the ***x***-axis of the TCP coordinate system.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

53.1 Syntax for MCGeo.Mechanism.Limitations

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Mechanism.Limitations.<Function or Access to
Value>(optional <Parameter 1, Parameter 2, Parameter n>);
```

53.2 Properties of MCGeo.Mechanism.Limitations

This function offers the following properties of the `Mechanism` function:

– WorkspaceLimitMin
– WorkspaceLimitMax
– OrientationLimitMin
– OrientationLimitMax
– OrientationFilter
– MaxPathVelocity

– MaxPathAcceleration
– MaxPathJerk
– SingularityDistance1

53.2.1 WorkspaceLimitMin

Description

The property reads or writes the smallest value for the workspace coordinates of the path group**.** The values are passed and read either individually for the composite coordinates X, Y, and Z, or via a variable of type `Position3D`.

property	data type	Meaning
WorkspaceLimitMin	Position3D	Minimum position values for the spatial coordinates in the workspace; absolute value relative to the reference point.
WorkspaceLimitMin.X	Double	The smallest position value for the X coordinate in the workspace.
WorkspaceLimitMin.Y	Double	The smallest position value for the Y coordinate in the workspace.
WorkspaceLimitMin.Z	Double	The smallest position value for the Z coordinate in the workspace.

Syntax

```
WorkspaceMin := geoGroup.Mechanism.Limitations.WorkspaceLimitMin;
geoGroup.Mechanism.Limitations.WorkspaceLimitMin := -100.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.2 WorkspaceLimitMax

Description

The property reads or writes the maximum value for the workspace coordinates of the path group. The values are passed and read either individually for the composite coordinates X, Y, and Z, or via a variable of type `Position3D`.

property	data type	Meaning
WorkspaceLimitMax	Position3D	Maximum position values for the spatial coordinates in the workspace; absolute value relative to the reference point.
WorkspaceLimitMax.X	Double	The largest position value for the X coordinate in the workspace.
WorkspaceLimitMax.Y	Double	The largest position value for the Y coordinate in the workspace.
WorkspaceLimitMax.Z	Double	The largest position value for the Z coordinate in the workspace.

Syntax

```
WorkspaceLimitMax := geoGroup.Mechanism.Limitations.WorkspaceLimitMax;
geoGroup.Mechanism.Limitations.WorkspaceLimitMax := 1000.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.3 OrientationLimitMin

Description

This property reads or writes the smallest value for the orientation angles in the path group's workspace. The values are either passed and read individually for the coordinate system orientations U, V, and W, or via a variable of type `Position3D`.

property	data type	Meaning
OrientationLimitMin	Position3D	Smallest set of orientation angles (Cardian angles U, V, W).
Orientation Limit Min. U	Double	Smallest orientation angle U.

property	data type	Meaning
OrientationLimitMin.V	Double	Smallest orientation angle V.
Orientation Limit Min. W	Double	Smallest orientation angle W.

Syntax

```
Orientation rangeMin := geoGroup.Mechanism.Limitations.OrientationLimitMin;
geoGroup.Mechanism.Limitations.OrientationLimitMin.U := 0.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.4 OrientationLimitMax

Description

The property reads or writes the largest value for the orientation angles in the path group's workspace. The values are either passed and read individually for the coordinate system orientations U, V, and W, or via a variable of type `Position3D`.

property	data type	Meaning
OrientationLimitMax	Position3D	Largest set of orientation angles (Cardian angles U, V, W).
OrientationLimitMax.U	Double	Maximum orientation angle U.
OrientationLimitMax.V	Double	Largest orientation angle V.
Orientation Limit Max. W	Double	Largest orientation angle W.

Syntax

```
Orientation rangeMax := geoGroup.Mechanism.Limitations.OrientationLimitMax;
geoGroup.Mechanism.Limitations.OrientationLimitMax.U := 360.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.5 OrientationFilter

Description

This property reads or writes the value for the orientation filter setting of the path group. Since orientation is not factored into the calculation of the trajectory, it is tracked as the path movement occurs. Particularly during over-grinding movements, a process without a filter would cause the orientation in the over-grinding area to jump. However, even in this case, the filter gently tracks the orientation.

property	data type	Meaning
OrientationFilter	Double	Filter value for orientation. Filter efficiency in %. 0% = no filtering effect; 100% = maximum filtering effect; Default: 70%; this setting should generally be sufficient.

Syntax

```
OrientationFilter := geoGroup.Mechanism.Limitations.OrientationFilter;
geoGroup.Mechanism.Limitations.OrientationFilter := 50.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.6 MaxPathVelocity

Description

data type	Meaning
Double	Reads or writes the value for the maximum path speed at which the TCP can or should be moved [in the user unit of speed]. This value may be determined, for example, by the machine's function or by the maximum specifications of the components involved in the drive system. The target speed of the path is limited to this value, even if the motion parameters would result in a higher speed.

Syntax

```
Maximum speed := geoGroup.Mechanism.Limitations.MaxPathVelocity;
geoGroup.Mechanism.Limitations.MaxPathVelocity := 200.5; // mm/s
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.7 MaxPathAcceleration

Description

data type	Meaning
Double	Reads or writes the value for the maximum path acceleration at which the TCP can or should be moved in the speed user unit. This value may be determined, for example, by the machine's operating characteristics or by the maximum specifications of the components involved in the drive system. The target acceleration and deceleration of the path are limited to this value, even if the motion parameters would otherwise result in a higher speed.

Syntax

```
MaxAccel := geoGroup.Mechanism.Limitations.MaxPathAcceleration;
geoGroup.Mechanism.Limitations.MaxPathAcceleration := 1300.0; // mm/s
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.8 MaxPathJerk

Description

data type	Meaning
Double	Reads or writes the value for the maximum path jerk with which the TCP can or should be moved in the acceleration unit. This value may be determined, for example, by the machine's operating characteristics or by the maximum specifications of the components involved in the drive system. The path driving torque is limited to this value, even if the motion parameters would result in a higher speed.

Syntax

```
MaxJerk := geoGroup.Mechanism.Limitations.MaxPathJerk;
geoGroup.Mechanism.Limitations.MaxPathJerk := 100000.0; // mm/s³
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

53.2.9 SingularityDistance1

In serial kinematics, a singularity occurs when the range of motion is restricted due to the positions of the individual joints. This is the case, for example, when a robot is fully extended. Before a singularity is reached, it may happen that a joint must reach a very high speed in order to keep up with the target speed of the TCP. That is why it makes sense not only to avoid these specific situations, but also to ensure that a certain distance is maintained from them. The MCX kernel therefore returns an error if the position of the TCP comes so close to a singular point that the distance becomes smaller than the threshold defined by the `SingularityDistance1` property.

Description

data type	Meaning
Double	Reads or writes the value for the singularity distance that must be maintained to prevent errors caused by reaching a singular position [in the user unit of the position].

Syntax

```
Singularity limit := geoGroup.Mechanism.Limitations.SingularityDistance1;
geoGroup.Mechanism.Limitations.SingularityDistance1 := 10.0;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

54 MCGeo.Mechanism.Slope

The **MCGeo.Mechanism.Slope** function allows you to read or adjust the calculated motion properties of the next motion block in a path group.

A positioning movement consists of the following phases:

1. acceleration slope
2. If applicable: Driving at a constant speed;

   The acceleration slope can also transition directly into the deceleration slope (triangular operation).
3. deceleration ramp

In addition, a transition polynomial can be used when two path sections are to merge seamlessly without stopping.

For path groups, the trajectory of the path speed is determined by a trapezoidal acceleration profile. The steepness of the slope in the trapezoid is determined by the specified maximum path jerk. In addition, the ac-
celeration and deceleration are the same along this section of path. Both are modified together via the `Acceleration` property.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

54.1 Syntax for MCGeo.Mechanism.Slope

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Mechanism.Slope.<Access to Value>
```

54.2 Properties of MCGeo.Mechanism.Slope

This function offers the following properties:

- SlopeState
- Is Accelerating
- IsAtConstantSpeed
- IsDecelerating
- IsStopped
- ActualMaxSpeed
- Maximum Acceleration
- SpeedSetpoint
- Acceleration Setpoint
- Speed
- SpeedOverride
- MaxVeloPoly
- Acceleration

54.2.1 SlopeState

Description

data type	Meaning
MCGeoSlopeStates	Reads the current phase of motion (acceleration, constant speed, deceleration, or stop) of the path setpoint for the path group. This allows you, for example, to activate an action in the machine only after the TCP has exited the acceleration phase.
GeoSlopeStates.Accelerating	The group is on the acceleration slope.
GeoSlopeStates.AtConstantSpeed	The group is traveling at a constant speed.
GeoSlopeStates.Decelerating	The group is on the deceleration slope.
GeoSlopeStates.Stopped	The group is down.

Syntax

```
SlopeState := geoGroup.Mechanism.Slope.SlopeState;
```

state administration

The property can be read in any state.

54.2.2 IsAccelerating

Description

data type	Meaning
Bool	Check whether the path setpoints are within the acceleration slope.

Syntax

```
If geoGroup.Mechanism.Slope.IsAccelerating , then …
```

state administration

The property can be queried in any state.

54.2.3 IsAtConstantSpeed

Description

data type	Meaning
Bool	Check whether the path setpoints are in the constant-speed phase.

Syntax

```
If geoGroup.Mechanism.Slope.IsAtConstantSpeed then …
```

state administration

The property can be queried in any state.

54.2.4 IsDecelerating

Description

data type	Meaning
Bool	Check whether the target values for the group are within the deceleration slope.

Syntax

```
If geoGroup.Mechanism.Slope.IsDecelerating , then …
```

state administration

The property can be queried in any state.

54.2.5 IsStopped

Description

data type	Meaning
Bool	Check whether the group target values are in the standstill phase.

Syntax

```
If geoGroup.Mechanism.Slope.IsStopped then …
```

state administration

The property can be queried in any state.

54.2.6 ActualMaxSpeed

Description

data type	Meaning
Double	Reads the value of the maximum path speed currently active in the group's target value generator. This value differs from the value of the Speed [▶ 303] property, for example, when movements are being processed in the pre-render buffer but the maximum speed value has since been changed.

Syntax

```
MaxSpeed := geoGroup.Mechanism.Slope.ActualMaxSpeed;
```

state administration

The property can be read in any state.

54.2.7 Maximum Acceleration

Description

data type	Meaning
Double	Reads the value of the maximum orbital acceleration currently active in the group's setpoint generator. This value differs from the value of the Acceleration [▶ 304] property, for example, when movements are being processed in the pre-buffer but the value for the maximum acceleration has since been changed.

Syntax

```
MaxAcceleration := geoGroup.Mechanism.Slope.ActualMaxAcceleration;
```

state administration

The property can be read in any state.

54.2.8 SpeedSetpoint

Description

data type	Meaning
Double	Reads the value of the target path velocity currently active in the MCX axis's target vale generator. The sign of the value is always positive.

Syntax

```
ActSpeedSetpoint := geoGroup.Mechanism.Slope.SpeedSetpoint;
```

state administration

The property can be read in any state.

54.2.9 Acceleration Setpoint

Description

data type	Meaning
Double	Reads the value of the setpoint acceleration currently active in the group's setpoint generator. The sign of the value indicates acceleration or deceleration.

Syntax

```
ActSetpoint := geo.Mechanism.Slope.AccelerationSetpoint;
```

state administration

The property can be read in any state.

54.2.10 Speed

Description

data type	Meaning
Double	Reads or writes the value for the maximum speed used to generate the path setpoint for a path movement. This value is typically passed as a parameter in the movement commands. The change in the value takes effect only after the preprocessing buffer has been processed.

Syntax

```
ProfileSpeed := geoGroup.Mechanism.Slope.Speed;
geoGroup.Mechanism.Slope.Speed := 300.0;
```

state administration

The property can be read and written in any state.

54.2.11 MaxVeloPoly

Description

data type	Meaning
Double	Reads or writes the value for the maximum speed used to generate the path setpoint for motion along interpolation polynomials. This causes the system to decelerate to this speed during the preceding motion block if the speed in that motion block was higher. If a value higher than the feed rate is programmed, the speed will be limited to the feed rate. The change in the value does not take effect until the preprocess buffer has been processed.

Syntax

```
UeberschleifGeschw := geoGroup.Mechanism.Slope.MaxVeloPoly;
geoGroup.Mechanism.Slope.MaxVeloPoly := 300.0;
```

state administration

The property can be read and written in any state.

54.2.12 SpeedOverride

Description

Speed override adjusts the target speed in the path setpoint generator by a certain percentage relative to the speed programmed in the motion command.

The maximum possible speed per path movement is determined by the *look-ahead* . The Look Ahead function calculates the path based on the programmed motion and the configured limits, and is therefore also linked to the programmed speed. This can result in a difference depending on whether the group is running, for example, at `a speed of 100 mm/s` with `a speed override of 150%` , or at `a speed of 150 mm/s` with `a speed override of 100%` ! To ensure the speed curve is as smooth as possible, it is therefore advisable to program the system using the maximum path speed and to set a value `of ≤ 100%` for `SpeedOverride` .

The target speed actually calculated by the interpolator is used as the basis for `SpeedOverride` . For example, if an effective jerk limit results in a target speed that is lower than the programmed speed, the `SpeedOverride` percentage is based on this lower target speed and not on the programmed speed.

data type	Meaning
Double	Reads or writes the value for speed override used to generate the setpoint for a path movement. This value is typically passed as a parameter in the movement commands. Changing the value takes effect immediately and applies to all types of path movement, provided the path movement is not currently in an acceleration slope or deceleration slope. `SpeedOverride` is generally limited to 200%.

Syntax

```
SpeedControl := geoGroup.Mechanism.Slope.SpeedOverride;
geoGroup.Mechanism.Slope.SpeedOverride := 50.0;
```

state administration

The property can be read and written in any state.

54.2.13 Acceleration

Description

data type	Meaning
Double	Reads or writes the value for the maximum acceleration used to generate the path set-point for a path movement. This value is typically passed as a parameter in the movement commands. The change in the value takes effect only after the preprocess buffer has been processed.

Syntax

```
ProfileAcceleration := geoGroup.Mechanism.Slope.Acceleration;
geoGroup.Mechanism.Slope.Acceleration := 5500.0;
```

state administration

The property can be read and written in any state.

55 MCGeo.Mechanism.Units

The **MCGeo.Mechanism.Units** function allows you to read or adjust the units used for position, velocity, acceleration, and jerk values.

These units are typically defined when the composite is created and can be queried using these commands in the STX program.

NOTICE



Changes in position due to a change of unit during machine operation

It is possible to change the units while the machine is running. However, since the positioning command contains only the values without specifying a unit, this changes the meaning of the values.

Example: Converting from millimeters to inches results in a different position in the machine for a value of `25.0` in the positioning command.

- ► Avoid changing units during operation.
- ► Be careful when converting units.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

55.1 Syntax for MCGeo.Mechanism.Units

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Mechanism.Units.<Function or Access to Value>(op-
tional <Parameter 1, Parameter 2, Parameter n>);
```

55.2 Functions of MCGeo.Mechanism.Units

This function offers the following functions:

– SetUnits()

55.2.1 SetUnits()

This function allows you to combine all units for position, velocity, acceleration, and jerk into a single function for path movement. You can use the parameters to assign different meanings to each unit.

transfer parameter

Parameters	data type	Meaning
PositionUnit	**MCLinearPositionUnits** (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for linear position values. Possible units: – millimeter – Customs – 1/10 millimeter – 1/100 millimeter
SpeedUnit	**MCLinearSpeedUnits** (optional) – MCLinearSpeedUnits.MM_PER_SEC – MCLinearSpeedUnits.MM_PER_MIN – MCLinearSpeedUnits.M_PER_SEC – MCLinearSpeedUnits.M_PER_MIN – MCLinearSpeedUnits.INCH_PER_SEC – MCLinearSpeedUnits.INCH_PER_MIN	Unit for linear velocity values. Possible units: – millimeters per second – millimeters per minute – meters per second – meters per minute – Inches per second – Inches per minute
AccelerationUnit	**MCLinearAccelerationUnits** (optional) – MCLinearAccelerationUnits.MM_PER_S2 – MCLinearAccelerationUnits.M_PER_S2 – MCLinearAcceleratio-nUnits.INCH_PER_S2	Unit for linear acceleration values. Possible units: – millimeters per second² – meters per second² – inches per second²
JerkUnit	**MCLinearJerkUnits** (optional) – MCLinearJerkUnits.MM_PER_S3 – MCLinearJerkUnits.M_PER_S3 – MCLinearJerkUnits.INCH_PER_S3	Unit for linear jerk values. Possible units: – millimeters per second³ – meters per second³ – Inches per second³
Orientation Unit	**MCGeoOrientationUnits** (optional) – MCGeoOrientationUnits.RPY_Angles	A leading to orientation. Possible units: – Gimbal angle (see MCGeo.Orientation)

Syntax

```
geoGroup.Mechanism.Units.SetUnits(
    MCLinearPositionUnits.MM,
    MCLinearSpeedUnits.MM_PER_SEC,
    MCLinearAccelerationUnits.MM_PER_S2,
    MCLinearJerkUnits.MM_PER_S3
    MCGeoOrientationUnits.RPY.Angles);
```

state administration

The function can be read in any state.

The function can be written in any state. However, it is recommended that you only modify units while they are in the **Locked** state.

55.3 Properties of MCGeo.Mechanism.Units

This function offers the following properties of the `Mechanism`function:

– PositionUnit
– Orientation Unit
– SpeedUnit
– AccelerationUnit
– JerkUnit

55.3.1 PositionUnit

Description

Reads or writes the unit in which the position values for a linearly defined axis are processed and displayed.

data type	Meaning
MCLinearPositionUnits (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for linear position values. Possible units: – millimeter – Customs – 1/10 millimeter – 1/100 millimeter

Syntax

```
PositionUnits := geoGroup.Mechanism.Units.PositionUnit;
geoGroup.Mechanism.Units.PositionUnit := MCLinearPositionUnits.MM_10TH;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

55.3.2 Orientation Unit

Description

Reads or writes the unit that processes and displays the orientation of the path movement.

data type	Meaning
MCGeoOrientationUnits (optional) – MCGeoOrientationUnits.RPY_Angles – MCGeoOrientationUnits.TCP_Vector	Unit to orientation. Possible units: – Gimballed angle (see MCGeo.Orientation [▶ 365])

Syntax

```
Ori_Units := geoGroup.Mechanism.Units.OrientationUnit;
geoGroup.Mechanism.Units.OrientationUnit :=
MCLinearOrientationUnits.RPY_Angles;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

55.3.3 SpeedUnit

Description

Reads or writes the unit in which the velocity values for path movement are processed and displayed.

data type	Meaning
MCLinearSpeedUnits (optional) – MCLinearSpeedUnits.MM_PER_SEC – MCLinearSpeedUnits.MM_PER_MIN – MCLinearSpeedUnits.M_PER_SEC – MCLinearSpeedUnits.M_PER_MIN – MCLinearSpeedUnits.INCH_PER_SEC – MCLinearSpeedUnits.INCH_PER_MIN	Unit for linear velocity values. Possible units: – millimeters per second – millimeters per minute – meters per second – meters per minute – Inches per second – Inches per minute

Syntax

```
SpeedUnits := geoGroup.Mechanism.Units.SpeedUnit;
geoGroup.Mechanism.Units SpeedUnit := MCLinearSpeedUnits.M_SEC;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

55.3.4 AccelerationUnit

Description

Reads or writes the unit in which the acceleration values for the path movement are processed and displayed.

data type	Meaning
MCLinearAccelerationUnits (optional) – MCLinearAccelerationUnits.MM_PER_S2 – MCLinearAccelerationUnits.M_PER_S2 – MCLinearAccelerationUnits.INCH_PER_S2	Unit for linear velocity values. Possible units: – millimeters per second² – meters per second² – inches per second²

Syntax

```
AccelerationUnits := geoGroup.Mechanism.Units.AccelerationUnit;
geoGroup.Mechanism.Units.AccelerationUnit :=
MCLinearAccelerationUnits.M_PER_S2;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

55.3.5 JerkUnit

Description

Reads or writes the unit in which the jerk values for the path movement are processed and displayed.

data type	Meaning
MCLinearJerkUnits (optional) – MCLinearJerkUnits.MM_PER_S3 – MCLinearJerkUnits.M_PER_S3 – MCLinearJerkUnits.INCH_PER_S3	Unit for linear velocity values. Possible units: – millimeters per second³ – meters per second³ – Inches per second³

Syntax

```
JerkUnits := geoGroup.Mechanism.Units.JerkUnit;
geoGroup.Mechanism.Units.JerkUnit := MCLinearJerkUnits.MM_PER_S3;
```

state administration

The property can be read and written in any state.

However, it is recommended that you only modify units while they are in the **Locked** state.

56 MCGeo Members

The **MCGeo.Members** function is primarily used to add synchronous axes to a path group or to remove them from it. Synchronous axes are axes that move in sync with the path movement without being mechanically connected to it. For a description of the axis types, see MCAxis.Mechanism.AxisMode [▶ 113].

For very specific cases, you can also add or remove composite axes from the STX program. Since this is recommended only in exceptional cases, the relevant commands are not described here. Please contact [Bucher Automation] Bucher Automation if you would like to use these features.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

56.1 Syntax for MCGeo.Members

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Members.<Function or Access to Value>(optional <Parameter>);
```

56.2 Features of MCGeo.Members

This function offers the following functions:

– AddSyncAxis()
– RemoveSyncAxis()

56.2.1 AddSyncAxis()

Use this function to add a synchronous axis to the path group. It is then set to the **"Inactive"** operating status and moved via the path group.

transfer parameter

Parameters	data type	Meaning
Name of the MCX axis	Pointer to MCAxis	Axis object that is to be moved in sync with the group.

Syntax

```
geoGroup.Members.AddSyncAxis(axSyncAxis);
```

state administration

To ensure that the functions are executed correctly, the path group and the synchronous axis must be in the same operating state. This can be either the **"Locked"** or **"Unlocked"** state.

Initial state	Allowed	Final state
Blocked	√	Error: Approved
Approved	√	Error: Approved

Initial state	Allowed	Final state
Working	X	Error: Approved
Interrupted	X	Error: Approved
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

The function can be called from any state of the master and slave axes.

56.2.2 RemoveSyncAxis()

This function removes a synchronous axis from the path group. The removed axis is set to the operating state that the path group had before the command was executed. This is either **Unlocked** or **Locked**.

transfer parameter

Parameters	data type	Meaning
Name of the MCX axis	Pointer to MCAxis	The axis object to be moved out of the group.

Syntax

```
geoGroup.Members.RemoveSyncAxis(axSyncAxis);
```

state administration

Initial state	Allowed	Final state
Blocked	√	Error: Approved
Approved	√	Error: Approved
Working	X	Error: Approved
Interrupted	X	Error: Approved
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

The function can be called from any state of the master and slave axes.

57 MCGeo.MoveCircle2D

The **MCGeo.MoveCircle2D** function allows you to initiate and configure a circular motion of the tool center point (TCP) in a spatial plane of an MCX path group. Linear motion can occur perpendicular to this plane, causing the TCP to describe a helical path.

Circular motion is defined by the center coordinates, the initial coordinates, and the angle of the circular segment. In addition, the length and direction of the movement in the orthogonal direction are specified.

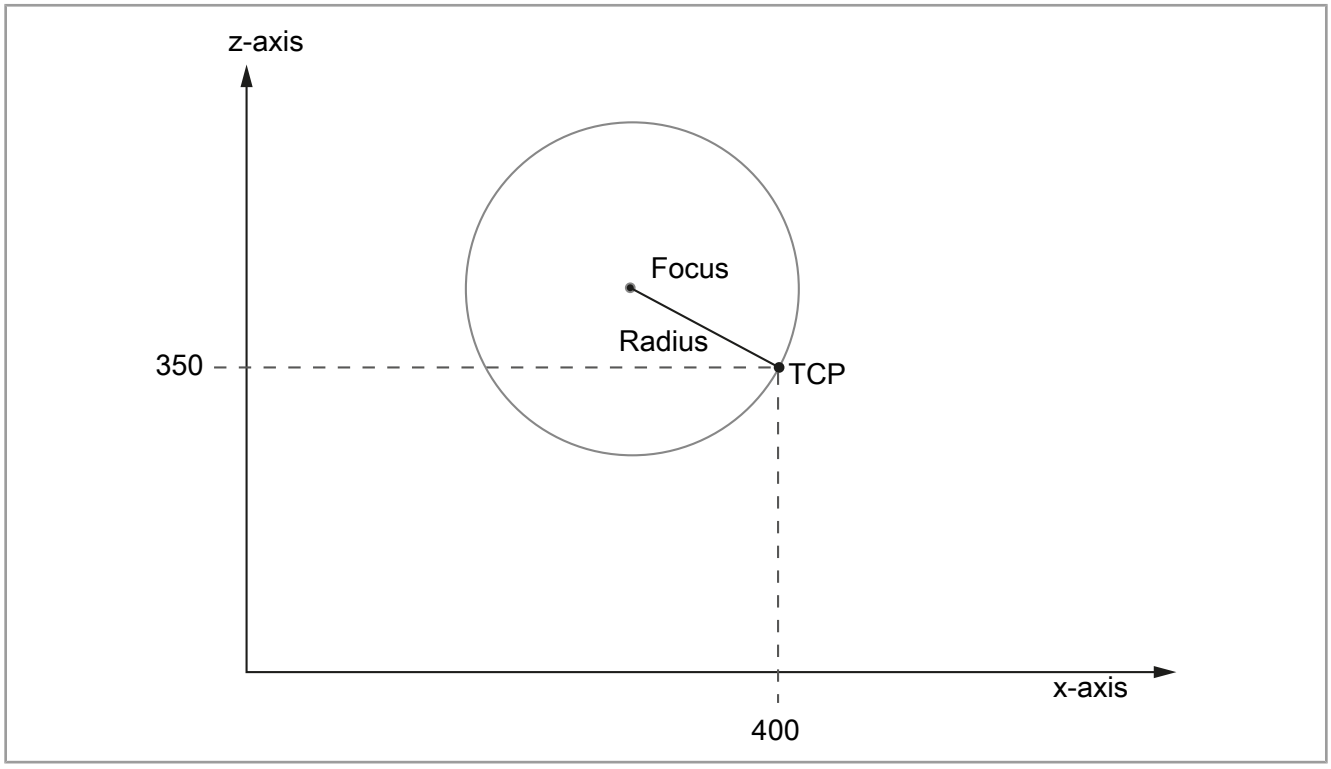


Fig. 11: Functions of MCGeo.MoveCircle2D

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes for procedures and sample projects on working with groups, particularly regarding loop-through operations.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

57.1 Syntax for MCGeo.MoveCircle2D

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveCircle2D.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

57.2 Functions of MCGeo.MoveCircle2D

This function offers the following functions:

– Start()

57.2.1 Start()

The function initiates a circular motion in a spatial plane of the TCP of a path group. The circular path is defined by:

– the coordinates of the target point of the last movement;
– the center coordinates;
– the angle to be painted over.

The position of the plane in the orthogonal direction (the starting point of the helical motion) is determined by the starting point (the destination coordinate of the last motion). If the center point coordinate in this direction differs from the coordinate value of the starting point, it is ignored.

Since a path group operates in buffered mode with a lead buffer, the first movement as a path segment begins immediately. If additional movement commands are issued during the movement, these segments are executed one after the other as soon as the previous segment has been completed. It is possible to connect path sections seamlessly by overlapping them.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.
– The parameters refer to the group object; that is, they constitute a property of the group. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode (see Positioning Mode [▶ 315]).
BlockEnd	MCGeoPathBlockEnd (optional)	Termination condition for a path section in the buffer. It determines when the next section of the path in the buffer begins. (see BlockEnd [▶ 316]).
Coordinate System Plane	MCGeoCoordinateSystemPlanes (optional)	The coordinate plane in which the arc is to be drawn.
	MCGeoCoordinateSystemPlanes.XY	The circular segment is moved in the XY plane.
	MCGeoCoordinateSystemPlanes.YZ	The circular segment is traced in the YZ plane.
	MCGeoCoordinateSystemPlanes.ZX	The circular segment is traced in the ZX plane.
TravelAngle	Double (optional)	The angle over which the arc should sweep, in [degrees]. The angle is swept in the specified plane in the mathematically positive direction.
TargetPositionHelixAxis	Double (optional)	Target position of the TCP at the coordinate that is orthogonal to the circular plane.
Speed	Double (optional)	Defines the path speed at which the TCP should move, provided that the motion is not in the slope phases (acceleration, deceleration). The value must be greater than zero.

Parameters	data type	Meaning
Smoothing Distance	Double (optional)	The value must always be less than half the length of the shorter of the two path sections that are to be connected by a crossover. The value must be greater than zero, even if the `BlockEnd` condition is set so that no smoothing occurs.
Delay	Integer (optional)	Defines the delay until the next movement stored in the buffer begins, in [ms]. The delay only takes effect if the positioning mode `InPosState_DestWnd_Delay` or `InPos-State_DestWnd_Delay_SyncOri` has been selected.
TargetWindow	Double (optional)	Defines the width of the target window relative to the motion path.
CenterPointXCenter-PointYCenterPointZ	Double (optional)	Three coordinate parameters for the center of the circular segment to be moved.
TargetOrientationU , TargetOrientationV, TargetOrientationW	Double (optional)	Three coordinate parameters for the tool's target position, which is to be reached at the end of the path segment.

Syntax

```
geoGroup.MoveCircle2D.Start(MCGeoPositioningModes.Absolute, MCGeoPathBlock-
End.InPosState, MCGeoCoordinateSystem-
Planes.XY,45.0,10.0,300.0,1.0,0,0.1,100.0,50.0,150.0,0.0,0.0,0.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	√		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various types of information are available to help you monitor path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or whether the group has been paused (MCGeo.State [▶ 376]).

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
mode and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **slope status** indicates which phase of motion the setpoint generator is currently in (MCGeo.Mechanism.Slope [▶ 299]).

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

57.3 Properties of MCGeo.MoveCircle2D

This function offers the following properties:

– PositioningMode
– BlockEnd
– Coordinate System Plane
– TravelAngle
– TargetPositionHelixAxis
– SmoothingDistance
– Delay
– TargetWindow
– CenterPoint
– TargetOrientation

57.3.1 PositioningMode

Description

This property allows you to read the currently active positioning mode of a path group or set a new value for that group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the final target position is calculated based on the actual position at the time of enabling.

data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolute	Moves the TCP to the position calculated from the travel angle, center point coordinates, and last target position coordinates. It is moved into position relative to the reference position. The direction of the path movement is determined by the sign of the parameter for the travel angle and the target helix position relative to the last target position coordinate in that direction. The orientation axes are controlled in such a way that the new orientation angles are reached via the shortest path. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis shifts 50° in the negative direction.
MCGeoPositioningModes.Relative	Specifies both the center coordinates relative to the starting point and the helix path relative to the plane at the starting point. The direction of the path movement is determined by the sign of the parameter for the travel angle and the target position of the helix.
MCGeoPositioningModes.AbsOriNormal	Moves the TCP's spatial coordinates in the same way as in `Absolute-` mode. Control the orientation axes so that the new orientation angles are achieved based on the difference between the last target angle and the desired angle. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis rotates 310° in the positive direction.

Syntax

```
Geo_PositioningMode := geoGroup.MoveCircle2D.PositioningMode;
geoGroup.MoveCircle2D.PositioningMode := MCGeoPositioningModes.Relative;
```

state administration

The property can be read and written in any state.

57.3.2 BlockEnd

Description

This property allows you to read the currently valid completion condition for a path segment in the lead buffer of the path group, or to set a new value for that group.

data type	Meaning
MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment will begin once the path segment defined here has reached the target speed `of 0` .
MCGeoPathBlockEnd.InPosState_DestWnd	The next path segment begins when the path's target position has reached the target position, minus the target window offset.

data type	Meaning
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment begins once the path's target position has been reached and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment begins once the path segment defined here has reached the target speed `of 0` and the setpoints of the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment will begin once the path's target position has reached the target position within the target window tolerance, and the setpoints of the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment begins once the path's target position has been reached and the dwell time has elapsed. In addition, the setpoints for the orientation axes must have reached their target values.
MCGeoPathBlockEnd.Smoothing	The next change of ends will take place without stopping. Depending on the `SmoothingDistance` parameter, a fifth-degree polynomial is calculated and inserted into the motion buffer as an interpolation term. This path-switching condition must not be used on the last path segment in the lead-in buffer.

Syntax

```
Geo_Satzuebergang := geoGroup.MoveCircle2D.BlockEnd;
geoGroup.MoveCircle2D.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

state administration

The property can be read and written in any state.

57.3.3 Coordinate System Plane

Description

This property allows you to read the currently valid value for the coordinate system plane of the 2D circular path segment in the lead-in buffer or of the most recently traversed 2D circular path segment in the path group. By editing the property, you set a new value for this group. This setting takes effect with the next `MoveCircle2D` command, unless a different value is specified there.

data type	Meaning
Type MCGeoCoordinateSystem-Planes (optional)	The coordinate plane in which the arc is to be drawn.
MCGeoCoordinateSystemPlanes.XY	The circular segment is moved in the XY plane.
MCGeoCoordinateSystemPlanes.YZ	The circular segment is traced in the YZ plane.
MCGeoCoordinateSystemPlanes.ZX	The circular segment is traced in the ZX plane.

Syntax

```
Segment plane := geoGroup.MoveCircle2D.CoordinateSystemPlane;
geoGroup.MoveCircle2D.CoordinateSystemPlane :=
MCGeoCoordinateSystemPlanes.ZX;
```

state administration

The property can be read and written in any state.

57.3.4 TravelAngle

Description

This property allows you to read the currently valid value for the arc angle of the path group or to specify a new value for the path group.

data type	Meaning
Double (optional)	Value for the angle of the circular segment that is drawn using `MoveCircle2D`.

Syntax

```
SegmentAngle := geoGroup.MoveCircle2D.TravelAngle;
geoGroup.MoveCircle2D.TraveöAmgöe := 45.2;
```

state administration

The property can be read and written in any state.

57.3.5 TargetPositionHelixAxis

Description

This property allows you to read the currently valid value for the target position of the helix coordinate of the path group or to specify a new value for the path group.

data type	Meaning
Double (optional)	Value for the target coordinate of the TCP in the direction of the orthogonal line to the coordinate plane of the circular segment.

Syntax

```
HelixPosition := geoGroup.MoveCircle2D.TargetPositionHelixAxis;
geoGroup.MoveCircle2D.TargetPositionHelixAxis := -250.0;
```

state administration

The property can be read and written in any state.

57.3.6 Smoothing Distance

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the path group transition distance when moving from one path segment to the next, or specifies a new value for the path group transition. The value must not be `0`, even if the `BlockEnd` condition is not `"smoothing"`.

Syntax

```
Smoothing distance := geoGroup.MoveCircle2D.SmoothingDistance;
geoGroup.MoveCircle2D.SmoothingDistance := 25.6;
```

state administration

The property can be read and written in any state.

57.3.7 Delay

Description

data type	Meaning
Integer (optional)	Reads the currently valid value for the wait time between two path segments in the path group's buffer, or specifies a new value for the path group in [ms]. This value takes effect only if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
WaitTime := geoGroup.MoveCircle2D.Delay;
geoGroup.MoveCircle2D.Delay := 150;
```

state administration

The property can be read and written in any state.

57.3.8 TargetWindow

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the target window setting of the path group, or writes a new value for that group. This is the distance from the target point at which the new path section begins. This value takes effect only if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri`, or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindow := geoGroup. MoveCircle2D.TargetWindow;
geoGroup.MoveCircle2D.TargetWindow := 0.7;
```

state administration

The property can be read and written in any state.

57.3.9 CenterPoint

Description

Use this property to read the current 3D coordinates of the circle center point for the current path movement of the path group, or to specify a new center for the group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
CenterPoint	Position3D (optional)	Center of the circle in 3D Cartesian coordinates.
CenterPoint.X	Position3D.X (Double)	X-coordinate of the circle center point.
CenterPoint.Y	Position3D.Y (Double)	Y-coordinate of the circle center point.
CenterPoint.Z	Position3D.Z (Double)	Z-coordinate of the circle center point.

Syntax

```
Center := geoGroup.MoveCircle2D.CenterPoint;
CenterPointX := geoGroup.MoveCircle2D.CenterPoint.X;
geoGroup.MoveCircle2D.CenterPoint := CenterPoint;
```

state administration

The property can be read and written in any state.

57.3.10 Target Orientation

Description

Use this property to read or specify the current 3D coordinate points of the target for the current path movement of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Orientation	Orientation3D (optional)	The goal of orienting the TCP in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value for the TCP orientation in the U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValueOri3D := geoGroup.MoveCircle2D.TargetOrientation;
TargetValueU := geoGroup.MoveCircle2D.TargetOrientation.U;
geoGroup.MoveCircle2D.TargetOrientation := NewOrientation;
```

state administration

The property can be read and written in any state.

58 MCGeo.MoveCircle3D

The **MCGeo.MoveCircle3D** function allows you to initiate and configure a circular motion of the Tool Center Point (TCP) of an MCX path group in any spatial orientation.

Circular motion is defined by three points in space that uniquely determine the arc:

– Starting point (final target point),
– Target point (end point of the circular segment motion),
– A point on the circle segment.

In the case of 2D kinematics, the Z-coordinates of the three points are ignored, and the arc is moved in the XY plane.

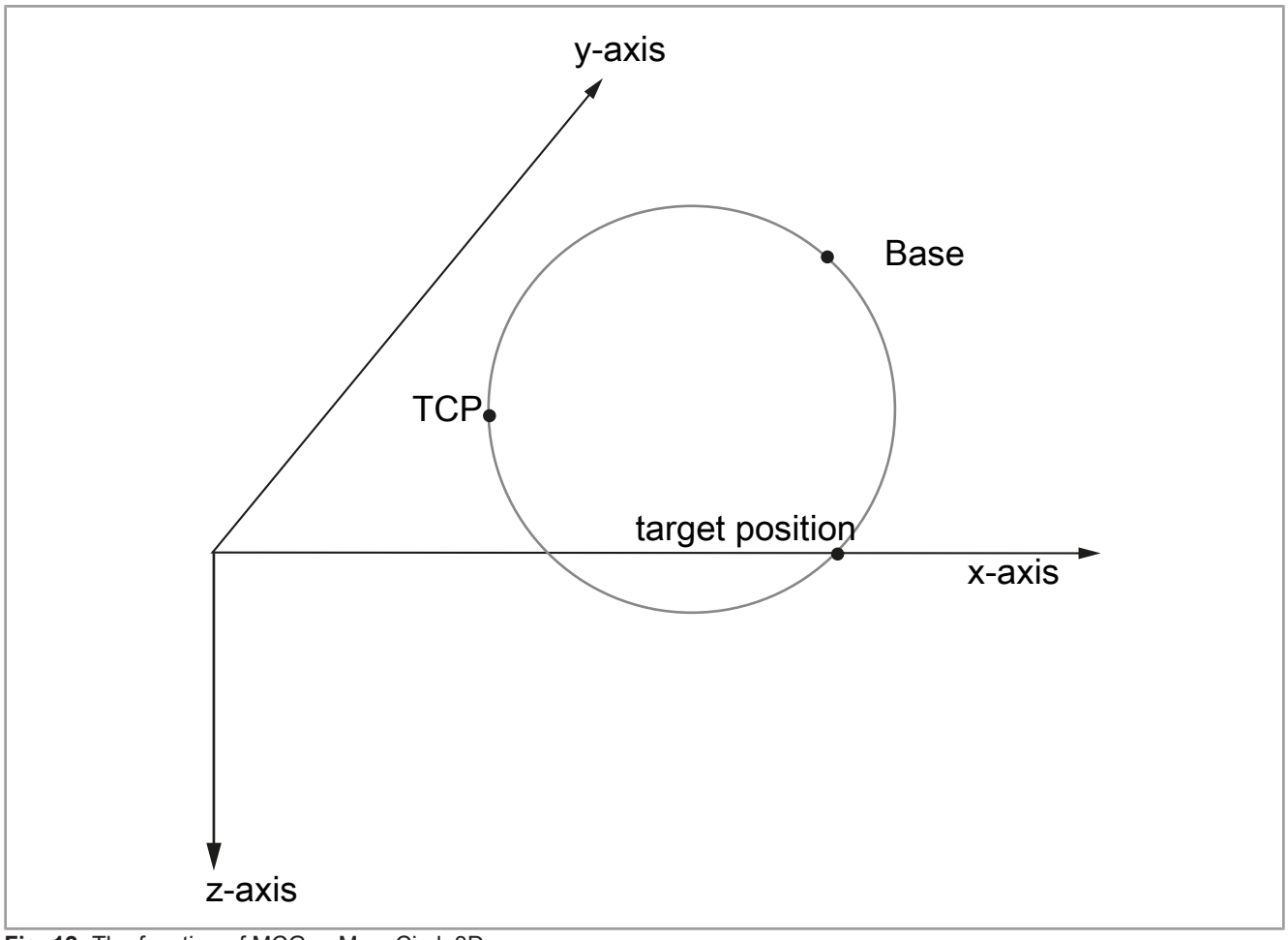


Fig. 12: The function of MCGeo.MoveCircle3D

A positioning movement consists of the following phases:

1. acceleration slope
2. If applicable: Driving at a constant speed;
   The acceleration slope can also transition directly into the deceleration slope (triangular operation).
3. deceleration ramp

In addition, a transition polynomial can be used when two path sections are to merge seamlessly without stopping.

For path groups, the trajectory of the vehicle's speed is determined by a trapezoidal acceleration profile. The steepness of the slope in the trapezoid is determined by the specified maximum path jerk. In addition, the ac-celeration and deceleration are the same along this section of path. Both are modified together using the `Acceleration` property.

Info

Further information

Please refer to the **Motion Control Application Notes or procedures and sample projects on working with groups, particularly regarding loop-through operations.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

58.1 Syntax for MCGeo.MoveCircle3D

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveCircle3D.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

58.2 Functions of MCGeo.MoveCircle3D

This function offers the following functions:

– Start()

58.2.1 Start()

The function initiates a circular motion of a path group's TCP in space. The circular path is defined by:

– the coordinates of the target point of the last movement;

– the coordinates of an intermediate point on the arc;

– the coordinates of a target point.

The position of the plane in the orthogonal direction (the starting point of the helical motion) is determined by the starting point (the destination coordinate of the last motion). If the center point coordinate in this direction differs from the coordinate value of the starting point, it is ignored.

Since a path group operates in buffered mode with a lead buffer, the first movement as a path segment begins immediately. If additional movement commands are issued during the movement, these segments are exe-cuted one after the other as soon as the previous segment has been completed. It is possible to connect path sections seamlessly by overlapping them.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the group object; that is, they constitute a property of the group. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode (see Positioning Mode [▶ 324]).
BlockEnd	MCGeoPathBlockEnd (optional)	Termination condition for a path section in the buffer. It determines when the next section of the path in the buffer begins (see BlockEnd [▶ 325]).
Speed	Double (optional)	Defines the path velocity at which the TCP should move, provided that the motion is not in the slope phases (acceleration, deceleration). The value must be greater than `0` .

Parameters	data type	Meaning
Smoothing Distance	Double (optional)	Overlap distance: The value must always be less than half the length of the shorter of the two path sections that are to be connected by a crossover. The value must be greater than `0` , even if the `BlockEnd` condition is set so that no looping occurs.
Delay	Integer (optional)	Defines the delay until the next movement stored in the buffer begins, in [ms]. The delay only takes effect if the positioning mode `InPosState_DestWnd_Delay` or `InPos-State_DestWnd_Delay_SyncOri` has been selected.
TargetWindow	Double (optional)	Defines the width of the target window relative to the motion path.
IntermediatePointXIn-termediatePointYInter-mediatePointZ	Double (optional)	Three coordinate parameters for the intermediate point on the arc to be traversed.
TargetPositionX , TargetPositionY, TargetPositionZ	Type: Double [optional]	Three coordinate parameters for the target point of the circular segment to be moved.
TargetOrientationU , TargetOrientationV, TargetOrientationW	Double (optional)	Three coordinate parameters for the tool's target position, which is to be reached at the end of the path segment.

Syntax

```
geoGroup.MoveCircle3D.Start(MCGeoPositioningModes.Absolute, MCGeoPathBlock-
End.InPosState,300.0,1.0,0,0.1,10.0,40.0,50.0,50.0,30.0,10.0,0.0,0.0,0.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	√		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various types of information are available to help you monitor path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or whether the group has been paused (MCGeo.State [▶ 376]).

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
mode and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **slope status** indicates which phase of motion the setpoint generator is currently in (MCGeo.Mechanism.Slope [▶ 299]).

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

58.3 Properties of MCGeo.MoveCircle3D

This function offers the following properties:

– PositioningMode
– BlockEnd
– SmoothingDistance
– Delay
– TargetWindow
– Intermediate Point
– TargetPosition
– Target Orientation

58.3.1 Positioning Mode

Description

This property allows you to read the currently active positioning mode of a path group or set a new value for that group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the final target position is calculated based on the actual position at the time of enabling.

data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolute	Moves the TCP to the position calculated from the parameters for the travel angle, center coordinates, and last target position coordinates. It is moved into position relative to the reference position. The direction of the path movement is determined by the sign of the parameter for the travel angle and the target helix position relative to the last target position coordinate in that direction. The orientation axes are controlled in such a way that the new orientation angles are reached via the shortest path. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis shifts 50° in the negative direction.
MCGeoPositioningModes.Relative	Specifies both the center coordinates relative to the starting point and the helix path relative to the plane at the starting point. The direction of the path movement is determined by the sign of the parameter for the travel angle and the target position of the helix.
MCGeoPositioningModes.AbsOriNormal	Moves the TCP's spatial coordinates in the same way as in `Absolute-` mode. Control the orientation axes so that the new orientation angles are achieved based on the difference between the last target angle and the desired angle. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis rotates 310° in the positive direction.

Syntax

```
Geo_PositioningMode := geoGroup.MoveCircle3D.PositioningMode;
geoGroup.MoveCircle3D.PositioningMode := MCGeoPositioningModes.Relative;
```

state administration

The property can be read and written in any state.

58.3.2 BlockEnd

Description

This property allows you to read the currently valid completion condition for a path segment in the lead buffer of the path group, or to set a new value for that group.

data type	Meaning
MCGeoPathBlockEnd (optional)	Termination condition for a path section in the buffer. It defines when the next section of the path in the buffer begins (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment will begin once the path segment defined here has reached the target speed `of 0` .
MCGeoPathBlockEnd.InPosState_DestWnd	The next path segment will begin once the path's target position has reached the target position, minus the target window offset.

data type	Meaning
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment begins once the path's target position has been reached and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment begins once the path segment defined here has reached the target speed `of 0` and the setpoints of the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment will begin once the path's current position has reached the target position, within the target window tolerance, and the setpoints for the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment begins once the path's target position has been reached and the dwell time has elapsed. In addition, the setpoints for the orientation axes must have reached their target values.
MCGeoPathBlockEnd.Smoothing	The next change of ends will take place without stopping. Depending on the `SmoothingDistance` parameter, a fifth-degree polynomial is calculated and inserted into the motion buffer as an interpolation term. This path-switching condition must not be used on the last path segment in the lead-in buffer.

Syntax

```
Geo_Satzuebergang := geoGroup.MoveCircle3D.BlockEnd;
geoGroup.MoveCircle3D.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

state administration

The property can be read and written in any state.

58.3.3 Smoothing Distance

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the path group transition distance when moving from one path segment to the next, or specifies a new value for the path group transition. The value must not be `0`, even if the `BlockEnd` condition is not `"smoothing"`.

Syntax

```
Smoothing distance := geoGroup.MoveCircle3D.SmoothingDistance;
geoGroup.MoveCircle3D.SmoothingDistance := 25.6;
```

state administration

The property can be read and written in any state.

58.3.4 Delay

Description

data type	Meaning
Integer (optional)	Reads the currently valid value for the wait time between two path segments in the path group's lead buffer, or specifies a new value for the path group in [ms]. This value takes effect only if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
WaitTime := geoGroup.MoveCircle3D.Delay;
geoGroup.MoveCircle3D.Delay := 150;
```

state administration

The property can be read and written in any state.

58.3.5 TargetWindow

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the target window setting of the path group, or writes a new value for that group. This is the distance from the target point at which the new path segment begins. This value takes effect only if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri`, or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindow := geoGroup. MoveCircle3D.TargetWindow;
geoGroup.MoveCircle3D.TargetWindow := 0.7;
```

state administration

The property can be read and written in any state.

58.3.6 Intermediate Point

Description

This property allows you to read the current 3D coordinates of the waypoint for the current path movement of the path group, or to specify a new waypoint for the group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
Intermediate Point	Position3D (optional)	Circle center point on the circle segment in 3D Cartesian coordinates.
IntermediatePoint.X	Position3D.X (Double)	X-coordinate of the midpoint.
IntermediatePoint.Y	Position3D.Y (Double)	Y-coordinate of the midpoint.
IntermediatePoint.Z	Position3D.Z (Double)	Z-coordinate of the intermediate point.

Syntax

```
IntermediatePoint := geoGroup.MoveCircle2D.IntermediatePoint;
IntermediatePointX := geoGroup.MoveCircle2D.IntermediatePoint.X;
geoGroup.MoveCircle2D.IntermediatePoint := IntermediatePoint;
```

state administration

The property can be read and written in any state.

58.3.7 TargetPosition

Description

Use this property to read the current 3D coordinates of the target for the current path movement of the path group, or to specify a new target point for the circular segment.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
TargetPosition	Position3D (optional)	The position of the TCP in 3D coordinates.
TargetPosition.X	Position3D.X (Double)	Target value for the TCP's position in the X direction.
TargetPosition.Y	Position3D.Y (Double)	Target value for the TCP's position in the Y direction.
TargetPosition.Z	Position3D.Z (Double)	Target value for the TCP's position in the Z direction.

Syntax

```
TargetValue3D := geoGroup.MoveCircle3D.TargetPosition;
TargetValueX := geoGroup.MoveCircle3D.TargetPosition.X;
geoGroup.MoveCircle3D.TargetPosition := NewTargetPoint;
```

state administration

The property can be read and written in any state.

58.3.8 Target Orientation

Description

Use this property to read or specify the current 3D coordinate values of the target for the current path movement of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Orientation	Orientation3D (optional)	The goal of orienting the TCP in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value for the TCP orientation in the U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValueOri3D := geoGroup.MoveCircle3D.TargetOrientation;
TargetValueU := geoGroup.MoveCircle3D.TargetOrientation.U;
geoGroup.MoveCircle3D.TargetOrientation := NewOrientation;
```

state administration

The property can be read and written in any state.

59 MCGeo.MoveLinear

The **MCGeo.MoveCircle3D** function allows you to initiate and configure a linear path movement for the tool center point (TCP) of an MCX path group.

A positioning movement consists of the following phases:

1. acceleration slope
2. If applicable: Driving at a constant speed;

   The acceleration slope can also transition directly into the deceleration slope (triangular operation).
3. deceleration slope

In addition, a transition polynomial can be used when two path sections are to merge seamlessly without stopping.

For path groups, the trajectory of the path speed is determined by a trapezoidal acceleration profile. The steepness of the slope in the trapezoid is determined by the specified maximum path jerk. In addition, the ac-
celeration and deceleration are the same along this section of path. Both are modified together via the `Acceleration` property.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) No. for procedures and sample projects on working with groups, particularly regarding loop-through operations.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

59.1 Syntax for MCGeo.MoveLinear

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveLinear.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

59.2 Functions of MCGeo.MoveLinear

This function offers the following functions:

– Start()

59.2.1 Start()

This function initiates a linear movement of the TCP in a path group.

Since a path group operates in buffered mode with a lead buffer, the first movement as a path segment begins immediately. If additional movement commands are issued during the movement, these segments are executed one after the other as soon as the previous segment has been completed. It is possible to connect path sections seamlessly by overlapping them.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the group object; that is, they constitute a property of the group. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode (see Positioning Mode [▶ 331]).
BlockEnd	MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It determines when the next section of the path in the lead-in buffer begins (see BlockEnd [▶ 332]).
Speed	Double (optional)	Defines the path speed at which the TCP should move, provided that the motion is not in a slope (ac-celeration or deceleration) phase. The value must be greater than `0` .
Smoothing Distance	Double (optional)	Overlap distance: The value must always be less than half the length of the shorter of the two path sections that are to be connected by a crossover. The value must be greater than `0` , even if the `BlockEnd`condition is set so that no looping occurs.
Delay	Integer (optional)	Defines the delay until the next movement stored in the buffer begins, in [ms]. The delay only takes effect if the positioning mode `InPosState_DestWnd_Delay` or `InPos-State_DestWnd_Delay_SyncOri` has been selected.
TargetWindow	Double (optional)	Defines the width of the target window relative to the motion path.
TargetPositionXTar-getPositionYTarget-PositionZ	Type: Double [optional]	Three coordinate parameters for the target position in space where the TCP should be located at the end of the path movement.
TargetOrientationU , TargetOrientationV, TargetOrientationW	Double (optional)	Three coordinate parameters for the tool's target position, which is to be reached at the end of the path segment.

Syntax

```
geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute, MCGeoPathBlock-
End.InPosState, 300.0,0.0,0,0.1,100.0,50.0,150.0,0.0,0.0,0.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	√		Interrupted
Error: Approved	X		Error: Approved

Initial state	Allowed	While running	Final state
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various types of information are available to help you monitor path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or whether the group has been paused (MCGeo.State [▶ 376]).

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **slope status** indicates which phase of motion the setpoint generator is currently in (MCGeo.Mechanism.Slope [▶ 299]).

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

59.3 Properties of MCGeo.MoveLinear

This function offers the following properties:

– Positioning Mode
– BlockEnd
– Smoothing Distance
– Delay
– TargetWindow
– Target Coordinates
– TargetPosition
– Target Orientation

59.3.1 PositioningMode

Description

This property allows you to read the currently active positioning mode of a path group or set a new value for that group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the final target position is calculated based on the actual position at the time of enabling.

data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolute	Moves the TCP to the position calculated from the parameters for the `TargetPosition` . It is moved into position relative to the reference position. The direction of the group´s path movement is determined by the last and new target positions. The orientation axes are controlled in such a way that the new orientation angles are reached via the shortest path. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis shifts 50° in the negative direction.
MCGeoPositioningModes.Relative	Moves the TCP forward by the distance specified in the `TargetPosition` parameter. The destination point is calculated by adding the coordinates of the last destination point to the value of the distances `TargetPosition`. The route coordinates can be specified with either a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	Moves the TCP's spatial coordinates in the same way as in `Absolute-` mode. Control the orientation axes so that the new orientation angles are achieved based on the difference between the last target angle and the desired angle. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis moves 310° in the positive direction.

Syntax

```
Geo_PositioningMode := geoGroup.MoveLinear.PositioningMode;
geoGroup.MoveLinear.PositioningMode := MCGeoPositioningModes.Relative;
```

state administration

The property can be read and written in any state.

59.3.2 BlockEnd

Description

This property allows you to read the currently valid completion condition for a path segment in the lead buffer of the path group, or to set a new value for that group.

data type	Meaning
MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment will begin once the path segment defined here has reached the target speed `of 0` .
MCGeoPathBlockEnd.InPosState_DestWnd	The next path segment will begin when the path's target position has reached the target position, minus the target window offset.

data type	Meaning
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment begins once the path's target position has been reached and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment begins when the path segment defined here has reached the target speed `of 0` and the setpoints of the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment will begin when the path's current position has reached the target position, within the target window tolerance, and the setpoints for the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment begins once the path's target position has been reached and the dwell time has elapsed. In addition, the setpoints for the orientation axes must have reached their target values.
MCGeoPathBlockEnd.Smoothing	The next change of ends will take place without stopping. Depending on the `SmoothingDistance` parameter, a fifth-degree polynomial is calculated and inserted into the motion buffer as an interpolation term. This path-switching condition must not be used on the last path segment in the lead-in buffer.

Syntax

```
Geo_Satzuebergang := geoGroup.MoveLinear.BlockEnd;
geoGroup.MoveLinear.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

state administration

The property can be read and written in any state.

59.3.3 SmoothingDistance

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the path group transition distance when moving from one path segment to the next, or specifies a new value for the path group transition. The value must not be `0`, even if the `BlockEnd` condition is not `"smoothing"`.

Syntax

```
Smoothing distance := geoGroup.MoveLinear.SmoothingDistance;
geoGroup.MoveLinear.SmoothingDistance := 25.6;
```

state administration

The property can be read and written in any state.

59.3.4 Delay

Description

data type	Meaning
Integer (optional)	Reads the currently valid value for the wait time between two path segments in the path group's lead buffer, or specifies a new value for the path group in [ms]. This value takes effect only if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
WaitTime := geoGroup.MoveLinear.Delay;
geoGroup.MoveLinear.Delay := 150;
```

state administration

The property can be read and written in any state.

59.3.5 TargetWindow

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the target window setting of the path group, or writes a new value for that group. This is the distance from the target point at which the new path segment begins. This value takes effect only if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri`, or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd`condition.

Syntax

```
TargetWindow := geoGroup. MoveLinear.TargetWindow;
geoGroup.MoveLinear.TargetWindow := 0.7;
```

state administration

The property can be read and written in any state.

59.3.6 Target Coordinates

Description

Use this property to read the current 6D coordinate points of the target for the current path movement of the path group, or to specify a new target point for the group.

The coordinate values are passed in a structure containing 3 directional coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Coordinates	Coordinate6D (optional)	The position of the TCP in 6D spatial coordinates.
TargetCoordinates.X	Coordinate6D.X (Double)	Target value for the TCP's position in the X direction.
TargetCoordinates.Y	Coordinate6D.Y (Double)	Target value for the TCP's position in the Y direction.
TargetCoordinates.Z	Coordinate6D.Z (Double)	Target value for the TCP's position in the Z direction.
TargetCoordinates.U	Coordinate6D.U (Double)	Target value for the TCP orientation in the U direction.
TargetCoordinates.V	Coordinate6D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetCoordinates.W	Coordinate6D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValue6D := geoGroup.MoveLinear.TargetCoordinates;
TargetY := geoGroup.MoveLinear.TargetCoordinates.Y;
geoGroup.MoveLinear.TargetCoordinates := NewTargetPoint;
```

state administration

The property can be read and written in any state.

59.3.7 TargetPosition

Description

Use this property to read the current 3D coordinates of the target for the current path movement of the path group, or to specify a new target point for the path group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
TargetPosition	Position3D (optional)	The position of the TCP in 3D coordinates.
TargetPosition.X	Position3D.X (Double)	Target value for the TCP's position in the X direction.
TargetPosition.Y	Position3D.Y (Double)	Target value for the TCP's position in the Y direction.
TargetPosition.Z	Position3D.Z (Double)	Target value for the TCP's position in the Z direction.

Syntax

```
TargetValue3D := geoGroup.MoveLinear.TargetPosition;
TargetValueX := geoGroup.MoveLinear.TargetPosition.X;
geoGroup.MoveLinear.TargetPosition := NewTargetPoint;
```

state administration

The property can be read and written in any state.

59.3.8 Target Orientation

Description

Use this property to read or specify the current 3D coordinate values of the target for the current path movement of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Orientation	Orientation3D (optional)	The goal of orienting the TCP in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value for the TCP orientation in the U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValueOri3D := geoGroup.MoveLinear.TargetOrientation;
TargetValueU := geoGroup.MoveLinear.TargetOrientation.U;
geoGroup.MoveLinear.TargetOrientation := NewOrientation;
```

state administration

The property can be read and written in any state.

60 MCGeo.MoveOrientation

The **MCGeo.MoveOrientation** function allows you to initiate and/or configure the movement of the orientation axes of the Tool Center Point (TCP) of an MCX path group.

Unlike the other `MCGeo.Move`commands, `MoveOrientation` only moves the orientation axes, while the spatial coordinate axes remain unchanged. It is not possible to override this action.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

60.1 Syntax for MCGeo.MoveOrientation

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveOrientation.<Function or Access to Value>(op-
tional <Parameter 1, Parameter 2, Parameter n>);
```

60.2 Functions of MCGeo.MoveOrientation

This function offers the following functions:

– Start()

60.2.1 Start()

This function initiates an orientation movement of the TCP for a path group. Since a path group operates in buffered mode with a lead buffer, the first movement as a path segment begins immediately. If additional movement commands are issued during the movement, these segments are executed one after the other as soon as the previous segment has been completed. It is possible to connect path sections seamlessly by overlapping them.

transfer parameter

Parameters	data type	Meaning
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode (see Positioning Mode [▶ 339]).
BlockEnd	MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).
Speed	Double (optional)	Defines the speed of movement in [°/s].
Acceleration	Double (optional)	Defines the acceleration of the motion in [°/s²].
Jerk	Double (optional)	Defines the jerk of the motion in [°/s³].
Delay	Integer (optional)	Defines the delay until the next movement stored in the buffer begins, in [ms]. The wait time only takes effect if the positioning mode `InPosState_DestWnd_Delay` has been selected.

Parameters	data type	Meaning
TargetWindow	Double (optional)	Defines the width of the target window relative to the path section [°].
TargetOrientationU , TargetOrientationV, TargetOrientationW	Double (optional)	Three coordinate parameters for the tool's target position, which is to be reached at the end of the path segment.

Syntax

```
geoGroup.MoveOrientation.Start(MCGeoPositioningModes.Absolute, MCGeoPath-
BlockEnd.InPosState,85.0,0.0,0.1,100.0,50.0,150.0,0.0,0.0,0.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	√		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various types of information are available to help you monitor path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or whether the group has been paused (MCGeo.State [▶ 376]).

```
When geoGroup.State.IsEnabled Continue; //The axis is in position control
mode and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **slope status** indicates which phase of motion the setpoint generator is currently in (MCGeo.Mechanism.Slope [▶ 299]).

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating , continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

60.3 Properties of MCGeo.MoveOrientation

This function offers the following properties:

– Positioning Mode

– BlockEnd

– Speed
– Acceleration
– Jerk
– Delay
– TargetWindow
– Target Orientation

60.3.1 Positioning Mode

Description

This property allows you to read the currently active positioning mode of a path group or set a new value for that group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the final target position is calculated based on the actual position at the time of enabling.

data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolute	Changes the TCP orientation to the values specified in the TargetOrientation parameter. It is moved to this position relative to the reference position. The orientation axes are controlled in such a way that the new orientation angles are reached via the shortest path. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis shifts 50° in the negative direction.
MCGeoPositioningModes.Relative	Adjusts the TCP orientation by the amount specified in the `TargetOrientation` parameter. Der Zielpunkt berechnet sich aus den Koordinaten des letzten Zielpunkts addiert mit dem Wert der Strecken in `TargetOrientation`. Values can be specified with either a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	Move the orientation axes so that the new orientation angles are achieved based on the difference between the last target angle and the desired angle. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis rotates 310° in the positive direction.

Syntax

```
Geo_PositioningMode := geoGroup.MoveOrientation.PositioningMode;
geoGroup.MoveOrientation.PositioningMode := MCGeoPositioningModes.Relative;
```

state administration

The property can be read and written in any state.

60.3.2 BlockEnd

Description

This property allows you to read the currently valid completion condition for a path segment in the lead buffer of the path group, or to set a new value for that group.

data type	Meaning
MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment will begin once the axis has reached a target speed of zero.
MCGeoPathBlockEnd.InPos-State_DestWnd	The next path segment begins when the target position of the axis with the longest travel distance has reached within the target window distance of the target position.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment begins once the target position of the axis with the longest travel distance has been reached and the dwell time has elapsed.

Syntax

```
Geo_Satzuebergang := geoGroup.MoveOrientation.BlockEnd;
geoGroup.MoveOrientation.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

state administration

The property can be read and written in any state.

60.3.3 Speed

Description

data type	Meaning
Double (optional)	Reads the current orientation speed value or sets a new value in [°/s]. This parameter defines the orientation speed at which the TCP should be moved, pro-vided that the movement is not currently in a slope (acceleration or deceleration). The value must be greater than zero.

Syntax

```
OriGeschw := geoGroup.MoveOrientation.Speed;
geoGroup.MoveOrientation.Speed := 15.4;
```

state administration

The property can be read and written in any state.

60.3.4 Acceleration

Description

data type	Meaning
Double (optional)	Reads the current value for orientation acceleration or sets a new value in [°/s²]. This parameter defines the orientation acceleration at which the TCP should be moved while the motion is in the ramps (acceleration, deceleration). The value must be greater than zero.

Syntax

```
OriRampe := geoGroup.MoveOrientation.Acceleration;
geoGroup.MoveOrientation.Acceleration := 120.1;
```

state administration

The property can be read and written in any state.

60.3.5 Jerk

Description

data type	Meaning
Double (optional)	Reads the current value for the orientation jerk or sets a new value in [°/s³]. This parameter defines the orientation jerk used to move the TCP while the motion is in the ramps (acceleration, deceleration). The jerk determines the steepness of the ramps. The value must be greater than zero.

Syntax

```
OriRuck := geoGroup.MoveOrientation.Jerk;
geoGroup.MoveOrientation.Jerk := 1514.4;
```

state administration

The property can be read and written in any state.

60.3.6 Delay

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the wait time between two path segments in the path group's lead buffer, or specifies a new value for the path group in [ms]. This value takes effect only if `InPosState_DestWnd_Delay` is used as the `BlockEnd`-condition.

Syntax

```
WaitTime := geoGroup.MoveOrientation.Delay;
geoGroup.MoveOrientation.Delay := 150;
```

state administration

The property can be read and written in any state.

60.3.7 TargetWindow

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the target angle offset from the orientation target point at which the new path segment begins, or writes a new value for this group. This value takes effect only if `InPosState_DestWnd` or `InPosState_DestWnd_Delay` is used as the `BlockEnd` condition.

Syntax

```
TargetWindow := geoGroup.MoveOrientation.TargetWindow;
geoGroup.MoveOrientation.TargetWindow := 0.7;
```

state administration

The property can be read and written in any state.

60.3.8 Target Orientation

Description

Use this property to read or specify the current 3D coordinate points of the target for the current path movement of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Orientation	Orientation3D (optional)	The goal of orienting the TCP in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value for the TCP orientation in the U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValueOri3D := geoGroup.MoveOrientation.TargetOrientation;
TargetValueU := geoGroup.MovePto.TargetOrientation.U;
geoGroup.MoveOrientation.TargetOrientation := NewOrientation;
```

state administration

The property can be read and written in any state.

61 MCGeo.MovePtp

The **MCGeo.MovePtp** function allows you to initiate and/or configure a point-to-point movement of the Tool Center Point (TCP) within the space of an MCX path group.

Unlike the other `MCGeo.Move`commands, `MovePtp` does not specify a fixed path. The axes of the path group are controlled so that they all reach their destination at the same time. The axis with the longest travel time determines the travel speed. The other axes are moved more slowly in proportion to their travel distance. The trajectory is determined by the kinematics and the starting and target positions in space. It is not possible to transition into or out of a PTP movement.

Movement using MovePtp is often the fastest way for the TCP to move within the space; it is used when the path taken is irrelevant to the function and when it is certain that the movement space is free of obstacles.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) No. for procedures and sample projects on working with groups, particularly regarding loop-through operations.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

61.1 Syntax for MCGeo.MovePtp

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MovePtp.<Function or Access to Value>(optional <Pa-
rameter 1, Parameter 2, Parameter n>);
```

61.2 Features of MCGeo.MovePtp

This function offers the following functions:

– Start()

61.2.1 Start()

This function initiates an orientation movement of the TCP for a path group. Since a path group operates in buffered mode with a lead buffer, the first movement as a path segment begins immediately. If additional movement commands are issued during the movement, these segments are executed one after the other as soon as the previous segment has been completed. It is possible to connect path sections seamlessly by overlapping them.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the group.

– The parameters refer to the group object; that is, they constitute a property of the group. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode (see Positioning Mode [▶ 345]).
BlockEnd	MCGeoMovePtpBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).

Parameters	data type	Meaning
Speed	Double (optional)	Defines the speed of the movement. It is expressed as a percentage of the maximum speed of the axis with the longest path distance. The other axes are moved more slowly in proportion to the length of their path range.
Delay	Integer (optional)	Defines the delay until the next movement stored in the buffer begins, in [ms]. The wait time only takes effect if the positioning mode `InPosState_DestWnd_Delay` has been selected.
TargetWindow	Double (optional)	Defines the width of the target window for the movement relative to the axis with the longest travel distance.
TargetPositionX , TargetPositionY, TargetPositionZ	Double (optional)	Three coordinate parameters for the target position in space where the TCP should be located after the path movement has been executed.
TargetOrientationU , TargetOrientationV, TargetOrientationW	Double (optional)	Three coordinate parameters for the tool's target position, which is to be reached at the end of the path segment.

Syntax

```
geoGroup.MovePtp.Start(MCGeoPositioningModes.Absolute, MCGeoPathBlockEnd.InPosState,85.0,0.0,0.1,100.0,50.0,150.0,0.0,0.0,0.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	√		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various types of information are available to help you monitor path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or whether the group has been paused (MCGeo.State [▶ 376]).

```
When geoGroup.State.IsEnabled Continue; //The group is in position control mode and is stationary.
```

```
When geoGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **slope status** indicates which phase of motion the setpoint generator is currently in (MCGeo.Mechanism.Slope [▶ 299]).

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

61.3 Properties of MCGeo.MovePtp

This function offers the following properties:

– Positioning Mode
– BlockEnd
– Delay
– TargetWindow
– TargetPosition
– Target Orientation
– Target Coordinates

61.3.1 PositioningMode

Description

This property allows you to read the currently active positioning mode of a path group or set a new value for that group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the final target position is calculated based on the actual position at the time of enabling.

data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolute	Changes the TCP orientation to the values specified in the `TargetOrientation` parameter. It is moved to this position relative to the reference position. The direction of movement for positioning is determined by the last position and the new target position. The orientation axes are controlled in such a way that the new orientation angles are reached via the shortest path. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis shifts 50° in the negative direction.

data type	Meaning
MCGeoPositioningModes.Relative	Adjusts the TCP orientation by the amount specified in the `TargetOrientation` parameter. The target point is calculated by adding the coordinates of the last target point to the value of the distances in `TargetOrientation`. Values can be specified with either a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	Move the orientation axes so that the new orientation angles are achieved based on the difference between the last target angle and the desired angle. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis moves 310° in the positive direction.

Syntax

```
Geo_PositioningMode := geoGroup.MovePtp.PositioningMode;
geoGroup.MovePtp.PositioningMode := MCGeoPositioningModes.Relative;
```

state administration

The property can be read and written in any state.

61.3.2 BlockEnd

Description

This property allows you to read the currently valid completion condition for a path segment in the lead buffer of the path group, or to set a new value for that group.

data type	Meaning
MCGeoMovePtpBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).
MCGeoMovePtpBlockEnd.InPosState	The next path segment will begin once the axis has reached a target speed of zero.
MCGeoMovePtpBlockEnd.InPosState_DestWnd	The next path segment begins when the target position of the axis with the longest path distance has reached within the target window distance of the target position.
MCGeoMovePtpBlockEnd.InPosState_DestWnd_Delay	The next path segment begins once the target position of the axis with the longest path distance has been reached and the dwell time has elapsed.

Syntax

```
Geo_Satzuebergang := geoGroup.MovePtp.BlockEnd;
geoGroup.MovePtp.BlockEnd := MCGeoMovePtpBlockEnd.InPosState_DestWnd;
```

state administration

The property can be read and written in any state.

61.3.3 Delay

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the wait time between two path segments in the path group's lead buffer, or specifies a new value for the path group in [ms]. This value takes effect only if `InPosState_DestWnd_Delay` is used as the `BlockEnd`-condition.

Syntax

```
WaitTime := geoGroup.MovePtp.Delay;
geoGroup.MovePtp.Delay := 150;
```

state administration

The property can be read and written in any state.

61.3.4 TargetWindow

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the target angle offset from the orientation target point at which the new path segment begins, or writes a new value for this group. This value takes effect only if `InPosState_DestWnd` or `InPosState_DestWnd_Delay` is used as the `BlockEnd` condition.

Syntax

```
TargetWindow := geoGroup.MovePtp.TargetWindow;
geoGroup.MovePtp.TargetWindow := 0.7;
```

state administration

The property can be read and written in any state.

61.3.5 TargetPosition

Description

Use this property to read the current 3D coordinates of the target for the current path movement of the path group, or to specify a new target point for the group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
TargetPosition	Position3D (optional)	The position of the TCP with 3D directional coordinates.
TargetPosition.X	Position3D.X (Double)	Target value for the TCP's position in the X direction.
TargetPosition.Y	Position3D.Y (Double)	Target value for the TCP's position in the Y direction.
TargetPosition.Z	Position3D.Z (Double)	Target value for the TCP's position in the Z direction.

Syntax

```
TargetValue3D := geoGroup.MovePtp.TargetPosition;
TargetValueX := geoGroup.MovePtp.TargetPosition.X;
geoGroup.MovePtp.TargetPosition := NewTargetPoint;
```

state administration

The property can be read and written in any state.

61.3.6 Target Orientation

Description

Use this property to read or specify the current 3D coordinate points of the target for the current path movement of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Orientation	Orientation3D (optional)	The goal of orienting the TCP in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value for the TCP orientation in the U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValueOri3D := geoGroup.MovePtp.TargetOrientation;
TargetValueU := geoGroup.MovePto.TargetOrientation.U;
geoGroup.MovePtp.TargetOrientation := NewOrientation;
```

state administration

The property can be read and written in any state.

61.3.7 Target Coordinates

Description

Use this property to read the current 6D coordinate points of the target for the current path movement of the path group, or to specify a new target point for the group.

The coordinate values are passed in a structure containing 3 directional coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

Target Orientation	data type	Meaning
Target Coordinates	Coordinate6D (optional)	The position of the TCP in 6D spatial coordinates.
TargetCoordinates.X	Coordinate6D.X (Double)	Target value for the TCP's position in the X direction.
TargetCoordinates.Y	Coordinate6D.Y (Double)	Target value for the TCP's position in the Y direction.

Target Orientation	data type	Meaning
TargetCoordinates.Z	Coordinate6D.Z (Double)	Target value for the TCP's position in the Z direction.
TargetCoordinates.U	Coordinate6D.U (Double)	Target value for the TCP orientation in the U direction.
TargetCoordinates.V	Coordinate6D.V (Double)	Target value for the TCP's orientation in the V direction.
TargetCoordinates.W	Coordinate6D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
TargetValue6D := geoGroup.MovePtp.TargetCoordinates;
TargetY := geoGroup.MovePtp.TargetCoordinates.Y;
geoGroup.MovePtp.TargetCoordinates := NewDestinationPoint;
```

state administration

The property can be read and written in any state.

62 MCGeo.MoveSpline

The **MCGeo.MoveSpline** function allows you to initiate or configure a spline-shaped path movement for the tool center point (TCP) of an MCX path group in space. In this type of interpolation, the path is approximated using cubic splines at up to 6 control points. Cubic splines are third-degree polynomials that are continuous up to the second derivative. The starting point (the last target point of the previous path command) is included as the first point in the polynomial calculation.

For more information, see the JetSymHelp section on splines.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN)**.These are available for download on the Bucher Automation website.

62.1 Syntax for MCGeo.MoveSpline

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveSpline.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

62.2 Functions of MCGeo.MoveCircle2D

This function offers the following functions:

– Start()

62.2.1 Start()

This function initiates a spline motion of the TCP for a path group. Since a path group operates in buffered mode with a lead buffer, the first movement as a path segment begins immediately. If additional movement commands are issued during the movement, these segments are executed one after the other as soon as the previous segment has been completed. It is possible to connect sections seamlessly by overlapping them.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the group.

– The parameters refer to the group object; that is, they constitute a property of the group. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Nodes	Integer (optional)	Number of control points to be used for the spline motion (minimum **2,** maximum 6).
Positioning Mode	MCGeoPositioningModes (optional)	Defines the positioning mode.
BlockEnd	MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition). See BlockEnd [▶ 354]

Parameters	data type	Meaning
Start Condition	Tyo MCGeoMoveSplineBlockTransitions (optional)	Defines the start behavior of the spline motion (see Start Condition [▶ 355]).
TargetCondition	Tyo MCGeoMoveSplineBlockTransitions (optional)	Defines the target behavior of the spline motion (see TargetCondition [▶ 356]).
Speed	Double (optional)	Defines the path speed at which the TCP should be positioned, provided that the motion is not in the slope phases (acceleration, deceleration). The value must be greater than zero.
Smoothing Distance	Double (optional)	Defines the overlap distance: The value must always be less than half the length of the shorter of the two path sections that are to be connected by a crossover. The value must be greater than zero, even if the `BlockEnd`condition is set so that no looping occurs.
Delay	**Integer** (optional)	Defines the delay in [ms] before the next movement in the buffer is initiated. The delay is only executed if the positioning mode `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` has been selected.
TargetWindow	Double (optional)	Defines the width of the target window relative to the motion path.
VirtualStartPositionX VirtualStartPositionY VirtualStartPositionZ	Double (optional)	Specifies a default starting position that is used for calculating the tangent when using the `ExplicitTangent` startup behavior. **NOTICE! Please note that, as a rule, the virtual starting position is not reached by the path. A tangent to the path is drawn at the first control point.**
VirtualTargetPositionX VirtualTargetPositionY VirtualTargetPositionZ	Double (optional)	Specifies a target position that is used to calculate the tangent for the `ExplicitTangent` target behavior. **NOTICE! Please note that, as a rule, the virtual target position is not reached by the path. A tangent to the path is drawn at the last con-trol point.**
Node0PositionX Node0PositionY Node0PositionZ	Double (optional)	Defines three coordinate parameters for the first control point in space, which are used to calculate the path movement.
Node0OrientationU Node0OrientationV Node0OrientationW	Double (optional)	Defines three coordinate parameters for the tool's orientation at the first waypoint, which are used to calculate the path.

Parameters	data type	Meaning
….		Coordinates and orientations for bases 1 … 4, similar to `Node0` and `Node5`.
Node5PositionX Node5PositionY Node5PositionZ	Double (optional)	Defines three coordinate parameters for the sixth control point in space, which are used to calculate the path movement.
Node5OrientationU Node5OrientationV Node5OrientationW	Double (optional)	Defines three coordinate parameters for the orientation of the tool at the sixth reference point, which are used to calculate the toolpath.

Syntax

```
geoGroup.MoveSpline.Start(5, //Number of nodes
MCGeoPositioningModes.Absolute, //PositioningMode
MCGeoPathBlockEnd.InPosState, //BlockEnd
MCGeoMoveSplineBlockTransitions.AutomaticMode, //StartCondition
MCGeoMoveSplineBlockTransitions.AutomaticMode, //TargetCondition
PathSpeed, //Speed
1.0, //Smoothing Distance
, // Delay - do not use with "inPosState"
0.5, //TargetWindow
,,, //VirtualStartPosition - do not use with "AutomaticMode"
,,, //VirtualTargetPosition - do not use with "AutomaticMode"
Node0X,Node0Y,Node0Z,Node0U,,, //Node 0 position and orientation
Node1X, Node1Y, Node1Z, Node1U, ... // Node 1 position and orientation
Node2X, Node2Y, Node2Z, Node2U, ... // Node 2 position and orientation
Node3X, Node3Y, Node3Z, Node3U,,, // Node 3 position and orientation
Node4X, Node4Y, Node4Z, Node4U); // Node 4 position and orientation
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	√		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various types of information are available to help you monitor path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or whether the group has been paused (MCGeo.State [▶ 376]).

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– The **slope status** indicates which phase of motion the setpoint generator is currently in (MCGeo.Mechanism.Slope [▶ 299]).

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

62.3 Properties of MCGeo.MoveSpline

This function offers the following properties:

– NodeCount
– Positioning Mode
– BlockEnd
– Start Condition
– TargetCondition
– Smoothing Distance
– Delay
– TargetWindow
– VirtualStartPosition
– VirtualTargetPosition
– NodePosition
– NodeOrientation
– NodeCoordinates

62.3.1 NodeCount

Description

data type	Meaning
Integer	Reads the currently valid value for the number of control points in a spline motion of the path group, or specifies a new value for the path group. Smallest value = 2, largest value = 6.

Syntax

```
SupportPointCount := geoGroup.MoveSpline.NodeCount;
geoGroup.MoveSpline.NodeCount := 6;
```

state administration

The property can be read and written in any state.

62.3.2 Positioning Mode

Description

This property allows you to read the currently active positioning mode of a path group or set a new value for that group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the final target position is calculated based on the actual position at the time of enabling.

data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolute	Moves the TCP to the position specified in the parameters for the last `NodeTargetPosition` . It is moved into position relative to the reference position. The direction of movement for positioning is determined by the last position and the new target position. The orientation axes are controlled in such a way that the new orientation angles are reached via the shortest path. The spatial coordinates of the TCP move in the same way as in `AbsOriNormal`mode. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis shifts 50° in the negative direction.
MCGeoPositioningModes.Relative	Moves the TCP along the path specified in the `NodeTargetPosition` parameters for each waypoint. Der Zielpunkt berechnet sich aus den Koordinaten des letzten Zielpunkts addiert mit dem Wert der Strecken in `NodeTargetPosition`. The route coordinates can be specified with either a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	The spatial coordinates of the TCP move in the same way as in `Absolute`mode. Move the orientation axes so that the new orientation angles are achieved based on the difference between the last target angle and the desired angle. **Example**: If the last target orientation was 40° and the new target orientation is 350°, the orientation axis moves 310° in the positive direction.

Syntax

```
Geo_PositioningMode := geoGroup.MoveSpline.PositioningMode;
geoGroup.MoveSpline.PositioningMode := MCGeoPositioningModes.Relative;
```

state administration

The property can be read and written in any state.

62.3.3 BlockEnd

Description

This property allows you to read the currently valid completion condition for a path segment in the lead buffer of the path group, or to set a new value for that group.

data type	Meaning
MCGeoPathBlockEnd (optional)	Termination condition for a path section in the lead-in buffer. It defines when the next section of the path in the lead-in buffer begins (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment will begin once the axis has reached a target speed of zero.
MCGeoPathBlockEnd.InPos-State_DestWnd	The next path segment begins when the target position of the axis with the longest travel distance has reached within the target window distance of the destination position.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment begins once the target position of the axis with the longest travel distance has been reached and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment begins once the path segment defined here has reached the target speed `of 0` and the setpoints for the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment will begin when the path's current position has reached the target position, within the target window tolerance, and the setpoints for the orientation axes have reached their target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment begins once the path's target position has been reached and the dwell time has elapsed. In addition, the setpoints for the orientation axes must have reached their target values.
MCGeoPathBlockEnd.Smoothing	The next change of ends will take place without stopping. Depending on the `SmoothingDistance` parameter, a fifth-degree polynomial is calculated and inserted into the motion buffer as a transition block.

Syntax

```
Geo_Satzuebergang := geoGroup.MoveSpline.BlockEnd;
geoGroup.MoveSpline.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

state administration

The property can be read and written in any state.

62.3.4 Start Condition

Description

This property allows you to read the currently active start behavior for the spline motion control or set a new value.

data type	Meaning
MCGeoMoveSplineBlockTransi-tions (optional)	Start behavior for the spline motion.
MCGeoMoveSplineBlockTransi-tions.ExplicitTangent	The *"Explicit Tangent Specification"* option requires a virtual starting position. This position is used in conjunction with the first waypoint to calculate the tangent direction along which the first waypoint is approached. Please note that, as a rule, the virtual starting position is not reached by the path. A tangent to the path is drawn at the first control point.

data type	Meaning
MCGeoMoveSplineBlockTransitions.AutomaticMode	In *automatic*mode, the transition from the previous sentence is calculated automatically. Typically, a parabolic path is used to connect the first three points. If no initial velocity is specified, the TCP is accelerated using the value set in the parameter for orbital acceleration.
MCGeoMoveSplineBlockTransitions.Tangential	The *"Tangential to Sub-path* " option behaves the same way as " `ExplicitTangent`," with the exception that the virtual start position is defined by the last target position of the sub-path and is approached within the sub-path.
MCGeoMoveSplineBlockTransitions.NaturalMode	In *natural mode* , the transition from the previous frame is calculated in such a way that the TCP does not experience any acceleration during the transition. This means that the transition occurs at a constant orbital velocity. If no initial velocity is specified, the TCP is accelerated using the value set in the path acceleration parameter.

Syntax

```
Geo_StartCondition := geoGroup.MoveSpline.StartCondition;
geoGroup.MoveSpline.StartCondition :=
MCGeoMoveSplineBlockTransitions.AutomaticMode;
```

state administration

The property can be read and written in any state.

62.3.5 TargetCondition

Description

This property allows you to read the currently active target behavior for the spline motion control or set a new value.

data type	Meaning
MCGeoMoveSplineBlockTransitions (optional)	Target behavior for the spline motion.
MCGeoMoveSplineBlockTransitions.ExplicitTangent	The *"Explicit Tangent"* option requires a virtual starting position. This position is used in conjunction with the first waypoint to calculate the tangent direction along which the first waypoint is approached. Please note that, as a rule, the virtual starting position is not reached by the path. A tangent to the path is drawn at the first control point.
MCGeoMoveSplineBlockTransitions.AutomaticMode	In *automatic* mode, the transition from the previous sentence is calculated automatically. Typically, a parabolic path is used to connect the first three points. If no initial velocity is specified, the TCP is accelerated using the value set in the parameter for orbital acceleration.
MCGeoMoveSplineBlockTransitions.Tangential	The *"Tangential* to Sub-path" option behaves the same way as "`ExplicitTangent`", with the exception that the virtual start position is defined by the last target position of the sub-path and is approached within the sub-path.
MCGeoMoveSplineBlockTransitions.NaturalMode	In *natural mode*, the transition from the previous frame is calculated in such a way that the TCP does not experience any acceleration during the transition. This means that the transition occurs at a constant orbital velocity. If no initial velocity is specified, the TCP is accelerated using the value set in the path acceleration parameter.

Syntax

```
Geo_TargetBehavior := geoGroup.MoveSpline.TargetCondition;
geoGroup.MoveSpline.TargetCondition :=
MCGeoMoveSplineBlockTransitions.Tangential;
```

state administration

The property can be read and written in any state.

62.3.6 Smoothing Distance

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the path group transition distance when moving from one path segment to the next, or specifies a new value for the path group transition. The value must not be `0`, even if the `BlockEnd` condition is not `"smoothing"`.

Syntax

```
Smoothing distance := geoGroup.MoveSpline.SmoothingDistance;
geoGroup.MoveLinear.SmoothingDistance := 25.6;
```

state administration

The property can be read and written in any state.

62.3.7 Delay

Description

data type	Meaning
Integer (optional)	Reads the currently valid value for the wait time between two path segments in the path group's lead buffer, or specifies a new value for the path group in [ms]. This value takes effect only if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
WaitTime := geoGroup.MoveSpline?.Delay;
geoGroup.MoveLinear.Delay := 150;
```

state administration

The property can be read and written in any state.

62.3.8 TargetWindow

Description

data type	Meaning
Double (optional)	Reads the currently valid value for the target window setting of the path group, or writes a new value for that group. This is the distance from the target point at which the new path section begins. This value takes effect only if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri`, or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindow := geoGroup. MoveSpline.TargetWindow;
geoGroup.MoveLinear.TargetWindow := 0.7;
```

state administration

The property can be read and written in any state.

62.3.9 VirtualStartPosition[x]

Description

Use this property to read the current 3D coordinates of the virtual starting position for the current spline motion of the path group, or to specify a new virtual starting position for the spline motion of the group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
VirtualStartPosition	Position3D (optional)	Position of the TCP's virtual starting point in 3D coordinates.
VirtualStartPosition.X	Position3D.X (Double)	Position for the virtual starting position of the TCP in the X direction.
VirtualStartPosition.Y	Position3D.Y (Double)	Position for the virtual starting position of the TCP in the Y direction.
VirtualStartPosition.Z	Position3D.Z (Double)	Position for the virtual starting position of the TCP in the Z direction.

Syntax

```
VirtStartPosition_3D := geoGroup.MoveSpline.VirtualStartPosition;
VirtStartPositionX := geoGroup.MoveSpline.VirtualStartPosition.X;
Geo.MoveSpline. VirtualStartPosition := VirtStartPos_3D;
```

state administration

The property can be read and written in any state.

62.3.10 VirtualTargetPosition[x]

Description

Use this property to read the current 3D coordinates of the virtual target position for the current spline motion of the path group, or to specify a new virtual target position for the spline motion of the group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
VirtualTargetPosition	Position3D (optional)	Position of the TCP's virtual target position in 3D directional coordinates.
VirtualTargetPosition.X	Position3D.X (Double)	Position of the TCP's virtual target position in the X direction.
VirtualTargetPosition.Y	Position3D.Y (Double)	Position of the TCP's virtual target position in the Y direction.
VirtualTargetPosition.Z	Position3D.Z (Double)	Position of the TCP's virtual target position in the Z direction.

Syntax

```
VirtTargetPosition_3D := geoGroup.MoveSpline.VirtualTargetPosition;
VirtTargetPositionX := geoGroup.MoveSpline.VirtualTargetPosition.X;
Geo.MoveSpline. VirtualTargetPosition := VirtTargetPos_3D;
```

state administration

The property can be read and written in any state.

62.3.11 NodePosition[n]

Description

Use this property to read the current 3D coordinate points of the control points for the current spline motion of the path group, or to specify a new target point for the group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
NodePosition[n]	Position3D (optional)	Positions of the TCP control points in 3D coordinates.
NodePosition[n].X	Position3D.X (Double)	Position of the TCP origin in the X-direction.
NodePosition[n].Y	Position3D.Y (Double)	Position of the TCP anchor point in the Y-direction.
NodePosition[n].Z	Position3D.Z (Double)	Position of the TCP base point in the Z direction.

Here , *[n]* is the number of the base.

Syntax

```
Stuetzppunkt3_3D := geoGroup.MoveSpline.NodePosition[3];
Stuetzpunkt3X := geoGroup.MoveSpline.NodePosition[3].X;
Geo.MoveSpline.NodePosition[2] := Pos2_3D;
```

state administration

The property can be read and written in any state.

62.3.12 NodeOrientation[n]

Description

Use this property to read the current 3D coordinate points of the control points for the current spline motion of the path group, or to specify a new target point for the group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
NodePosition[n]	Position3D (optional)	The target of the orientation for the TCP control points in 3D orientation coordinates.
NodePosition[n].U	Position3D.U (Double)	Orientation of the TCP base in the U direction.
NodePosition[n].V	Position3D.V (Double)	Orientation of the TCP anchor point in the V direction.
NodePosition[n].W	Position3D.W (Double)	Orientation of the TCP base point in the west-east direction.

Syntax

```
StuetzpunktOri2_3D := geoGroup.MoveSpline.NodeOrientation[2];
StuetzpunktOU := geoGroup.MoveSpline.NodeOrientation[0].U;
geoGroup.MoveSpline.NodeOrientation[3] := Orientation3_3D;
```

state administration

The property can be read and written in any state.

62.3.13 NodeCoordinates[n]

Description

Use this property to read the current 6D coordinate points of the control points for the current spline motion of the path group, or to specify a new control point for the group.

The coordinate values are passed in a structure containing 3 directional coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
NodeCoordinates[n]	Coordinate6D (optional)	TCP base point coordinates in 6D space coordinates.
NodeCoordinates[n].X	Coordinate6D.X (Double)	Position of the TCP origin in the X direction.
NodeCoordinates[n].Y	Coordinate6D.Y (Double)	Position of the TCP anchor point in the Y-direction.
NodeCoordinates[n].Z	Coordinate6D.Z (Double)	Position of the TCP base point in the Z direction.
NodeCoordinates[n].U	Coordinate6D.U (Double)	Orientation of the TCP anchor points in the U direction.
NodeCoordinates[n].V	Coordinate6D.V (Double)	Orientation of the TCP anchor points in the V direction.
NodeCoordinates[n].W	Coordinate6D.W (Double)	Orientation of the TCP anchor points in the west-east direction.

Here *, [n]* is the number of the base.

Syntax

```
Stuetzpunkt2_6D := geoGroup.MoveSpline.NodeCoordinates[2];
Stuetzpunkt0Y := geoGroup.MoveSpline.NodeCoordinates[0].Y;
geoGroup.MoveSpline.NodeCoordinates[4] := ControlPoint4_6D;
```

state administration

The property can be read and written in any state.

63 MCGeo.MoveSyncAxis

The **MCGeo.MoveSyncAxis** function allows you to retrieve information from the synchronous axes of a path group and assign motion parameters to the synchronous axes so that they move in the desired manner along with the path group.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) No. for procedures and sample projects on working with synchronous axes in path groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

63.1 Syntax for MCGeo.MoveSyncAxis

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveSyncAxis.<Function or Access to Value>(optional
<Parameter>);
```

63.2 Functions of MCGeo.MoveSyncAxis

This function offers the following functions:

- Start()
- GetTargetPosition()
- SetTargetPosition()
- GetPositioningMode()
- SetPositioningMode()

63.2.1 Start()

This function adds a new motion for the synchronous axis to the path buffer. The movement starts in sync with the next path movements in the path group. The synchronized motion is distributed across the path movements in such a way that the target position (`TargetPosition` parameter) is reached when the path group stops with the `BlockEnd`parameter `set to InPosState` . If, after the `Start()` command for the synchronous axis, multiple path segments are written to the path buffer and concatenated without stopping (`BlockEnd` parameter set to `"Smoothing"`), the synchronous axis will move through these path segments and will not reach the target position until the next path segment stops with the `BlockEnd` parameter set to `"InPosState"`.

transfer parameter

Parameters	data type	Meaning
Name of the MCAxis	Pointer to MCAxis	Axis object that is to be moved in sync with the group. This axis must first be added to the path group using AddSyncAxis() [▶ 310] .
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode for synchronized movement.
	MCGeoPositioning-Modes.Absolute	The synchronous axis will move to the position specified in the `TargetPosition` parameter with the next path movement. It is moved to this position relative to the reference position. The direction of movement for the positioning is determined by the last and the new target positions.
	MCGeoPositioning-Modes.Relative	The synchronous axis will move forward by the distance specified in the `TargetPosition` parameter with the next path movement. The target position is calculated by adding the value in `TargetPosition` to the last target position. The direction of travel depends on the sign of the `TargetPosition` parameter.
TargetPosition	Double (optional)	Sets the target position for the synchronized movement.

Syntax

```
geoGroup.MoveSyncAxis.Start(AxSync, MCGeoPositioningModes.Absolute,300.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	√	Approved
Working	√	Approved
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

63.2.2 GetTargetPosition()

transfer parameter

Parameters	data type	Meaning
Name of the MCAxis	Pointer to MCAxis	Reads the most recently provided value for the target position of the synchronous axis. This axis must first be added to the path group using AddSyncAxis() [▶ 310] .

Syntax

```
TargetPos := geoGroup.MoveSyncAxis.GetTargetPosition(AxSync);
```

state administration

The function can be read in any state.

63.2.3 SetTargetPosition()

transfer parameter

Parameters	data type	Meaning
Name of the MCAxis	Pointer to MCAxis	Sets a new value for the target position of the synchronous axis. This axis must first be added to the path group using AddSyncAxis() [▶ 310] . This value is stored only in the axis object. It only takes effect when called with a Start() [▶ 361]command that does not specify a target position.

Syntax

```
geoGroup.MoveSyncAxis.SetTargetPosition(AxSync,400.0);
```

state administration

The following table applies when writing the function:

63.2.4 GetPositioningMode()

transfer parameter

Parameters	data type	Meaning
Name of the MCAxis	Pointer to MCAxis	Reads the most recently provided value for the positioning mode of the synchronous axis. The axis must first be added to the path group using AddSyncAxis() [▶ 310] .

Syntax

```
PosMod := geoGroup.MoveSyncAxis.GetPositioningMode(AxSync);
```

state administration

The function can be read in any state.

63.2.5 SetPositioningMode()

transfer parameter

Parameters	data type	Meaning
Name of the MCAxis	Pointer to MCAxis	Axis object that is to be moved in sync with the group. This axis must first be added to the path group using AddSyncAxis() [▶ 310] .
Positioning Mode	MCGeoPositioningModes (optional)	Sets the positioning mode for synchronized movement.
	MCGeoPositioning-Modes.Absolute	The synchronous axis moves to the position specified in the `TargetPosition` parameter with the next path movement. It is moved to this position relative to the reference position. The direction of movement for positioning is determined by the last and the new target positions.
	MCGeoPositioning-Modes.Relative	The synchronous axis will move forward by the distance specified in the `TargetPosition` parameter with the next path movement. The target position is calculated by adding the value in `TargetPosition` to the last target position. The direction of travel depends on the sign of the `TargetPosition` parameter.

Syntax

```
geoGroup.MoveSyncAxis.SetPositioningMode(AxSync, MCGeoPositioningModes.Abso-
lute);
```

state administration

The following table applies when writing the function:

64 MCGeo.Orientation

The **MCGeo.Orientation** function allows you to retrieve the current values for 3D orientation, sorted by coordinates, for kinematics that have one or more orientation axes.

The orientation of the Tool Center Point (TCP) is defined by the so-called **Cardan angles**, in the axis order ***z – y' – x"***, relative to the base coordinate system.

Calculation of the gimbal angle

To derive the TCP coordinate system ***(x''', y''', z''')*** from the base coordinate system (x, y, z), the following procedure is used:

1. First, the fixed coordinate system is rotated by an angle U around the ***z***-axis. This results in the coordinate system ***(x', y', z')***.
2. This is now rotated by an angle V about the ***y***'-axis, resulting in the coordinate system ***(x'', y'', z'')*** .
3. Finally, this coordinate system is rotated by an angle W around the ***x***''-axis. This defines the TCP coordinate system ***(x''', y''', z''')***.

Meaning of the gimbal angle

– The angle U (also known as the yaw angle) describes the orientation of the TCP relative to the ***x-y***plane of the base coordinate system.

– The angle V (also known as the pitch angle) describes the orientation of the TCP relative to the ***y-z***plane of the Cartesian coordinate system.

– The angle W (also known as the roll angle) describes the rotation of the TCP around the ***x***-axis of the TCP coordinate system.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

64.1 Syntax for MCGeo.Orientation

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Orientation.<Access to Value>;
```

64.2 Properties of MCGeo.Orientation

This function offers the following properties:

– Actual

– Setpoint

– Target

64.2.1 Actual

Description

This property returns the current 3D coordinate points of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
Actual	Orientation3D	Actual value of the TCP orientation in 3D orientation coordinates.
Actual.U	Orientation3D.U (Double)	Current value of the TCP orientation in the U direction.
Actual.V	Orientation3D.V (Double)	Current value of the TCP orientation in the V direction.
Actual.W	Orientation3D.W (Double)	Current value of the TCP's orientation in the W-direction.

Syntax

```
IstOri3D := geoGroup.Orientation.Actual;
IstOriU := geoGroup.Orientation.Actual.U;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

64.2.2 Setpoint

Description

Use this property to read the current 3D coordinate points of the path group's setpoint generator.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
Setpoint	Orientation3D	Target value for the TCP orientation in 3D orientation coordinates.
Setpoint.U	Orientation3D.U (Double)	Setpoint for the TCP orientation in the U direction.
Setpoint.V	Orientation3D.V (Double)	Setpoint for the TCP's orientation in the V direction.
Setpoint.W	Orientation3D.W (Double)	Target value for the TCP's orientation in the W direction.

Syntax

```
SetpointOri3D := geoGroup.Orientation.Setpoint;
SetpointU := geoGroup.Orientation.Setpoint.U;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

64.2.3 Target

Description

Use this property to read the current 3D coordinate points of the target of the current path movement of the path group.

The coordinate values are passed in a structure containing 3 orientation coordinates (U, V, W). Only the values that are relevant to the kinematics of the path group are passed on. The other values remain zero.

property	data type	Meaning
Target	Orientation3D	The goal of orienting the TCP in 3D orientation coordinates.
Target.U	Orientation3D.U (Double)	The goal is to orient the TCP in the U direction.
Target.V	Orientation3D.V (Double)	The goal is to align the TCP in the V direction.
Target.W	Orientation3D.W (Double)	The goal is to orient the TCP in the W direction.

Syntax

```
TargetValueOri3D := geoGroup.Orientation.Target;
TargetValueU := geoGroup.Orientation.Target.U;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

65 MCGeo.Position

The **MCGeo.Orientation** function allows you to retrieve the current values for 3D path movement, sorted by coordinates.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

65.1 Syntax for MCGeo.Position

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Position.<Access to Value>;
```

65.2 Properties of MCGeo.Position

This function offers the following properties:

– Actual
– Setpoint
– Target

65.2.1 Actual

Description

This property returns the current 3D coordinate points of the path group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
Actual	Position3D	Actual value of the TCP's position in 3D coordinates.
Actual.X	Position3D.X (Double)	Current value of the TCP position in the X direction
Actual.Y	Position3D.Y (Double)	Current value of the TCP position in the Y direction
Actual.Z	Position3D.Z (Double)	Current value of the TCP position in the Z direction

Syntax

```
Ist3D := geoGroup.Position.Actual;
X := geoGroup.Position.Actual.X;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

65.2.2 Setpoint

Description

Use this property to read the current 3D coordinate points of the path group's setpoint generator.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
Setpoint	Position3D	Desired value for the TCP's position in 3D coordinates.
Setpoint.X	Position3D.X (Double)	Setpoint for the TCP position in the X direction.
Setpoint.Y	Position3D.Y (Double)	Target value for the TCP's position in the Y direction.
Setpoint.Z	Position3D.Z (Double)	Setpoint for the TCP position in the Z direction.

Syntax

```
Setpoint3D := geoGroup.Position.Setpoint;
SetpointX := geoGroup.Position.Setpoint.X;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

65.2.3 Target

Description

Use this property to read the current 3D coordinates of the target of the current path movement within the path group.

The coordinate values are passed in a structure with three directional coordinates (X, Y, Z). Only the values that are relevant to the kinematics of the path group are passed. The other values remain zero.

property	data type	Meaning
Target	Position3D	The position of the TCP in 3D coordinates.
Target.X	Position3D.X (Double)	The X-coordinate of the TCP's position.
Target.Y	Position3D.Y (Double)	The Y-coordinate of the TCP's position.
Target.Z	Position3D.Z (Double)	The position of the TCP in the Z direction.

Syntax

```
TargetValue3D := geoGroup.Position.Target;
TargetValueX := geoGroup.Position.Target.X;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

66 MCGeo.Power

The **MCGeo.Power** function allows you to enable or disable the control—and thus the torque generation—of a path group. All axes involved in the group are switched to the new operating state simultaneously.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

66.1 Syntax for MCGeo.Power

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Power.<Function>(optional <Parameter>;
```

66.2 Features of MCGeo.Power

This function offers the following functions:

– Enable()
– Disable()
– Quickstop()

66.2.1 Enable()

This function activates the controllers of the servo amplifiers for all coupled axes and applies power to the mo-tor. A path group can only be activated if all path group axes are operational.

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.Power.Enable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Approved
Approved	–	Approved
Working	–	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Therefore, check whether the new state has been reached in the program before calling any further commands for the group.

```
When geoGroup.State.IsEnabled , continue;
```

66.2.2 Disable()

This function disables the servo amplifier of all coupled axes for torque generation (pulse inhibit). The motor will immediately lose power and coast to a stop if no brake is applied.

transfer parameter

Parameters	data type	Meaning
DisableModes	MCPowerDisableMode (optional)	Mode for executing the `Disable()` function.
	MCPowerDisableMode.Forced (Default)	The pulse block takes effect immediately in all cases. This mode is also active if no parameter is specified.
	MCPowerDisableMode.StateCheck	The pulse inhibit function is only active when the group is in the **"Enabled"** operating state. In all other states, an operating error is displayed.

Syntax

```
geoGroup.Power.Disable();
geoGroup.Power.Disable(MCPowerDisableModes.Forced);
geoGroup.Power.Disable(MCPowerDisableModes.StateCheck);
```

state administration

The following table applies when writing the function:

The following table applies when the function is called without parameters or with the parameter `` `MCPowerDisableModes.Forced` ``:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	√	Blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The following table applies to calls using the `MCPowerDisableModes.StateCheck` parameter:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked

Initial state	Allowed	Final state
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before executing any further commands for the group, check whether the new state has been reached in the program as follows:

```
If geoGroup.State.IsDisabled , continue;
```

If the axis was previously in the **"Error: Enabled"** state, use the following command.

```
When geoGroup.State.IsDisabledError Continue;
```

66.2.3 QuickStop()

This function stops all coupled axes using the quick-stop deceleration and locks the servo amplifiers once they have come to a stop.

NOTICE



No position control of the coupled axes during a quick stop

During the rapid stop delay, there is no position control, and therefore no coordinated movement of the coupled axes in accordance with the path movement. Each coupled axis comes to a stop at a different time, depending on its current speed and the quick-stop deceleration setting.

► Use this command only in emergencies!

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.Power.QuickStop();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	√	Blocked

Initial state	Allowed	Final state
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before executing any further commands for the group, check whether the new state has been reached in the program as follows:

```
If geoGroup.State.IsDisabled , continue;
```

66.3 Example task: MCGeo.Power

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample 1 : Autorun
    //Unlock the coupled axes
    If geoGroup.State.IsDisabled Then
        geoGroup.Power.Enable();
    End_if;
    When geoGroup.State.IsEnabled , continue;

    //A process may be associated with the group here, or other tasks may
move the group
    //…

    // Lock the group
    If geoGroup.State.IsRunning Then    //if the group is still moving, stop
after a reasonable delay

        geoGroup.MoveHalt.Start();
    When geoGroup.Mechanism.Slope.IsStopped , continue;

    End_if;
    geoGroup.Power.Disable(); // Disable the group
    If geoGroup.State.IsDisabled , continue;

End_Task;
```

67 MCGeo.Settings

The **MCGeo.Settings** function allows you to adjust or save all parameters of an MCX path group at once.

Do not adjust parameters while a save operation is in progress
Saving the parameters to a file is a process that runs in the background. In other words, it spans several cycles of the MCX. If parameter values are changed while the data is being saved, it is impossible to predict whether the old or new value will be written to the file.

How Parameter Management Works

Parameters saved in JetSym 's motion setup are stored in a text file on the PC and, if requested, also in the controller's file system Motion subfolder. The filename of the parameter file on the controller is *MCGeo_X.ini*, where *X* stands for the path group ID number. The path group ID is displayed on the Properties page in the MCX group's motion setup. In addition, the tooltip in the Hardware tree displays the path group ID when the mouse pointer hovers over the group.

When the controller boots up, the MCX kernel reads the data from the *MCGeo_X.ini*file and uses it for the relevant group. If such a file is present on the controller, no further parameterization is required. If no corresponding file exists for the group, the parameters are loaded with default values.

Normally, no modifications to the file are required from within the STX program.

In very rare cases, however, it may be advisable to reload the parameter file onto the controller or to save the parameters. However, when saving, it is important to note that the parameter file saved in the JetSym project may no longer match the parameters currently in effect on the machine.

67.1 Syntax for MCGeo.Settings

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Settings.<Function>(optional <Parameter>;
```

67.2 Features of MCGeo.Settings

This function offers the following functions:

– SaveToFile()
– LoadFromFile()

67.2.1 SaveToFile()

transfer parameter

Parameters	data type	Meaning
Suffix	Integer (optional)	Enables saving the parameters of the MCX group to a file in the "Motion" subfolder within the controller's file system. Depending on the transfer parameter, different files are created or overwritten: – Parameter `0` or no parameter: The *MCGeo_X.ini* parameter file for this group will be created or overwritten with the current values. – Parameter not equal to `0`: The *MCGeo_X_TmpY.ini* parameter file is created or overwritten with the current values. In this case , *Y* is equal to the parameter `suffix`.

Syntax

```
geoGroup.Settings.SaveToFile();    // Overwrites (or creates) the MC-
Geo_1.ini file (if geoGroup has Geo-ID 1)
geoGroup.Settings.SaveToFile(3);    // Creates the file MCGeo_1_Tmp3.ini (if
geoGroup has Geo-ID 1)
```

state administration

The function can be called from any state.

67.2.2 LoadFromFile()

transfer parameter

Parameters	data type	Description
Suffix	Integer (optional)	Enables the import of MCX group parameters from a file located in the "Motion" subfolder within the controller's file system. Depending on the transfer parameter, different files are read. – Parameter `0` or no parameter: The data from the *MCGeo_X.ini* file is read and used for the relevant group. – Parameter not equal to `0`: The data from the *MCGeo_TmpY.ini* file is read and used for the relevant group. In this case , *Y* is equal to the parameter `suffix`.

Syntax

```
geoGroup.Settings.LoadFromFile();    //Reads the MCGeo_1.ini file (if ge-
oGroup has Geo-ID 1)
geoGroup.Settings.LoadFromFile(3);    //Reads the file MCGeo_1_Tmp3.ini (if
geoGroup has Geo-ID 1)
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

68 MCGeo.State

The **MCGeo.State** function allows you to retrieve various details about a path group, such as operating state, movement type, and so on.

Basic and additional information

In this context, please also note the following information:

– For state transitions, see the section MCGeo.State.Transitions [▶ 387];
– For a detailed description of the MCX core architecture, see the JetSymonline help
– For basic information, see Motion Control Application Note No. 54 (MCX Motion Control Basics).

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

operating state

The operating state (`OperatingMode`) specifies the general way in which the path group operates. There are 3 operating states:

operating state	Meaning
`Inactive`	The group cannot accept commands, for example, because it has not been activated.
`Normal` (normal)	The group is usable. Traverse commands are executed in a buffered manner (see MCGeo.Buffer [▶ 256]). In other words, all movement commands are stored in a buffer and processed one after another.
`Error`	An operator error has occurred. The operating state is retained; however, no new functions or parameters can be passed.

Tab. 6: Operating states of MCX path groups

Operating status

At any given time, a group is in one, and only one, clearly defined operating state (`OperatingState`). For example, if the group is in the **"Running"**operating state, the **"Enabled"** operating state is not active, even though all group axes naturally have pulse enable.

In a given state, only a subset of all function calls is permitted. The state model for the axis is as follows:

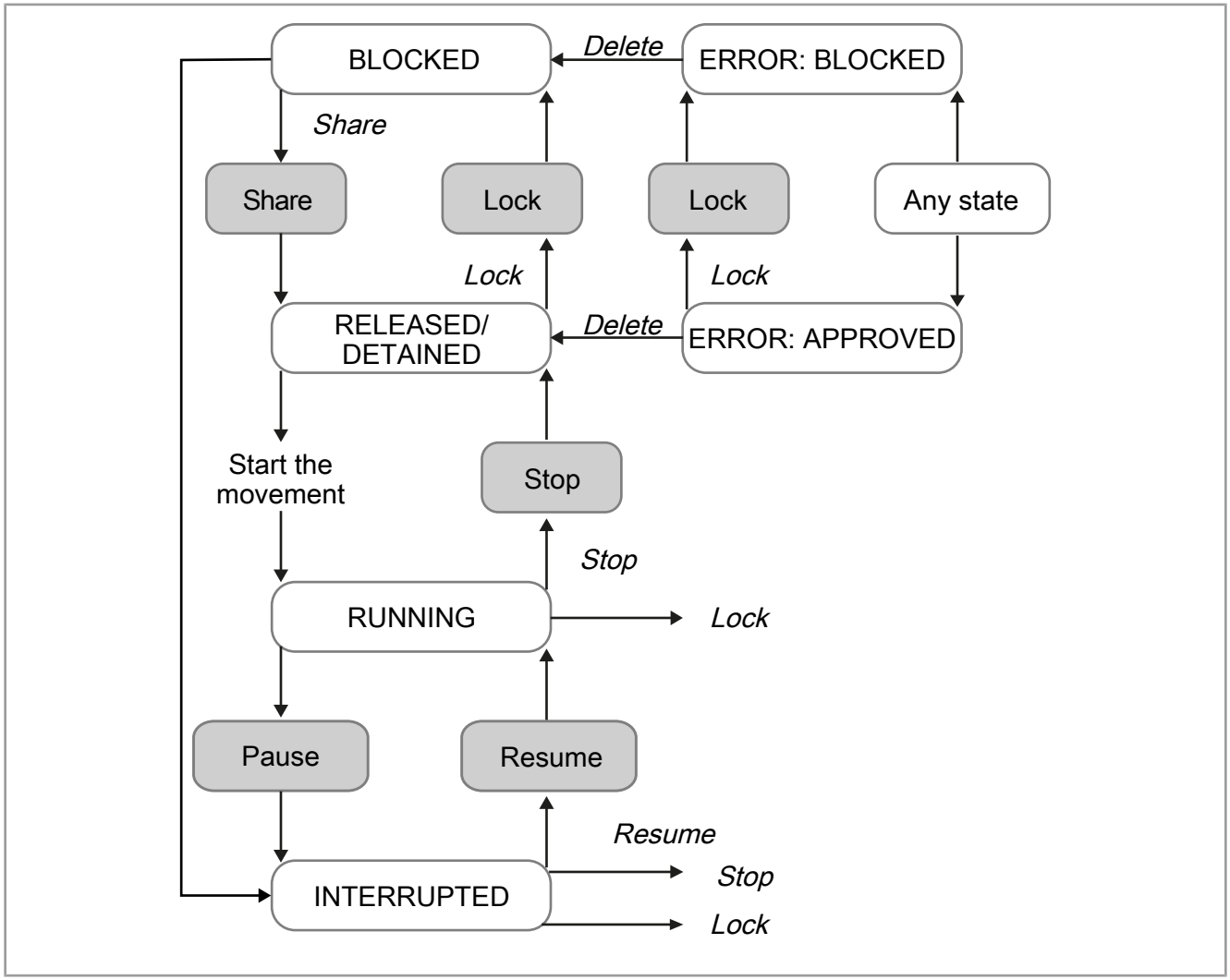


Fig. 13: State-space model for MCX groups

Operating status	Meaning
Inactive	The group cannot accept commands, for example, because it has not been activated.
Disabled	– The servo amplifier controllers for all coupled axes are open (no pulse enable). – No torque is generated.
Enabled	– The controllers for all coupled axes are closed (pulse enable). – The setpoint interpolator is at a standstill.
Running (In progress)	At least one axis is in motion; that is, the setpoint interpolator transmits motion setpoints to the servo amplifier.
Suspended	– This corresponds to the **"Enabled"** operating state, with the exception that the target position is retained. A movement that has been paused using `MCGeo.State.Transitions.Suspend()` can be resumed using the command `MCGeo.State.Transitions.Resume()`.
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control mode (error level 4).
DisabledError (Error: Locked)	An error has occurred. After the error response, control of all axes is disabled (error levels 5 and 6).

Tab. 7: Operating statuses of path groups

Type of movement

The motion type (`TypeOfMotion`) specifies the type of motion that the setpoint generator is currently generating for the path group.

Querying the transaction type is useful, for example, in a monitoring task where the process is unknown. Based on the active motion type prior to shutdown, it is possible to control the restart correctly.

Operating status	Meaning
Not a move	This transaction type remains active until the first transaction in the group is executed.
Straight (`MoveLinear`)	The path group is currently moving in a straight line, or was moving in a straight line just a moment ago.
Circular Motion 2D (`MoveCircular2D`)	The path group is currently performing, or most recently performed, a 2D circular motion.
Circular Motion 3D (`MoveCircular3D`)	The path group is currently performing, or was most recently performing, a 3D circular motion.
Spline (`MoveSpline`)	The path group is currently executing, or most recently executed, a spline motion.
Polynomial (`MovePolynomial`)	The path group is undergoing a restructuring.
Point to Point (`MovePtp`)	The path group is currently moving, or was most recently moving, from point A to point B.
Orientation (`MoveOrientation`)	The path group is currently performing, or has recently performed, a turning maneuver.
Linked movement (`MoveExternal`)	The path group is currently, or was most recently, operated using a linked movement.

68.1 Properties of MCGeo.State

This function offers the following properties:

- Value
- IsDisabled
- IsEnabled
- IsRunning
- IsSuspended
- IsDisabledError
- IsEnabledError
- IsInactive
- Operating Mode
- TypeOfMotion
- Transitions (for a description, see MCGeo.State.Transitions [▶ 387])

68.1.1 Value

Description

data type	Meaning
Operating States	Reads the operational status of the path group (for a detailed description, see the section *"Operational Status"* in MCGeo.State [▶ 376]).
OperatingStates.Inactive	Inactive
OperatingStates.Disabled	Blocked
OperatingStates.Enabled	Approved
OperatingStates.Running	Working
OperatingStates.Suspended	Interrupted
OperatingStates.EnabledError	Error: Approved
OperatingStates.DisabledError	Error: blocked

Syntax

```
OpState := geoGroup.State.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.2 IsDisabled

Description

data type	Meaning
Bool	Check whether the group is in the **"Locked"** state.

Syntax

```
If geoGroup.State.IsDisabled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.3 IsDisabledError

Description

data type	Meaning
Bool	Check whether the group is in the **"Error: Locked"** state.

Syntax

```
When geoGroup.State.IsDisabledError , Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.4 IsEnabled

Description

data type	Meaning
Bool	Checks whether the group is in the **"Enabled"** state.

Syntax

```
When geoGroup.State.IsEnabled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.5 IsEnabledError

Description

data type	Meaning
Bool	Check whether the group is in the **"Error: Enabled"** operational state.

Syntax

```
When geoGroup.State.IsEnabledError , Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.6 IsInactive

Description

data type	Meaning
Bool	Check whether the group is in the **inactive** state.

Syntax

```
If geoGroup.State.IsInactive , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.7 IsRunning

Description

data type	Meaning
Bool	Check whether the group is in the **"Running"** state.

Syntax

```
When geoGroup.State.IsRunning , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.8 IsSuspended

Description

data type	Meaning
Bool	Check whether the group is in the **"Disconnected"** state.

Syntax

```
When geoGroup.State.IsSuspended , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.9 OperatingMode.Value

Description

data type	Meaning
Type MCOperatingModes	Reads the operating state of the group (for a detailed description, see the "Operating State" [▶ 376]section).
MCOperatingModes.Inactive	The group is inactive, for example, because it has not been activated.
MCOperatingModes.Normal	The group has been activated and is not in an error state.
MCOperatingModes.Error	The group is in an error state following an operator error.

Syntax

```
OpModeAxis := geoGroup.State.OperatingMode.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInInactiveMode

Description

data type	Meaning
Bool	Check whether the axis is in the *inactive* operating state.

Syntax

```
When geoGroup.State.OperatingMode.IsInInactiveMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInErrorMode

Description

data type	Meaning
Bool	Check the operating state of the system to see if it is in *error* mode.

Syntax

```
If geoGroup.State.OperatingMode.IsInErrorMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInNormalMode

Description

data type	Meaning
Bool	Check the operating state of the group.

Syntax

```
When geoGroup.State.OperatingMode.IsInNormalMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.1.10 TypeOfMotion.Value

Description

Operating status	Meaning
Type MCGeoTypesOfMotion	The movement type currently assigned to the path group (for a detailed description, see the section MCGeo.State [▶ 378]").
MCGeoTypesOfMotion.MoveLinear	The path group is currently moving in a straight line, or was moving in a straight line just a moment ago.
MCGeoTypesOfMotion.MoveCircular2D	The path group is currently performing, or most recently performed, a 2D circular motion.
MCGeoTypesOfMotion.MoveCircular3D	The path group is currently performing, or was most recently performing, a 3D circular motion.
MCGeoTypesOfMotion.MoveSpline	The path group is currently executing, or most recently executed, a spline motion.
MCGeoTypesOfMotion.MovePolynomial	The path group is undergoing a restructuring.
MCGeoTypesOfMotion.MovePtp	The path group is currently, or was most recently, traveling from one point to another.
MCGeoTypesOfMotion.MoveOrientation	The path group is currently performing, or has recently performed, a turning maneuver.
MCGeoTypesOfMotion.MoveExternal	The path group is currently being operated, or was most recently operated, using a synchronized movement.

Syntax

```
TypeOfMotion := geoGroup.State. TypeOfMotion.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveLinear

Description

data type	Meaning
Bool	Checks whether the group is in the `"Linear"` motion mode.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveLinear Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveCircular2D

Description

data type	Meaning
Bool	Checks whether the group is currently performing a `2D circular motion` or has most recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveCircular2D Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveCircular3D

Description

data type	Meaning
Bool	Checks whether the group is currently performing a `3D circular motion` or has most recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveCircular3D Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveSpline

Description

data type	Meaning
Bool	Checks whether the group is currently performing a `spline motion` or has most recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveSpline , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMovePolynomial

Description

data type	Meaning
Bool	Checks whether the group is currently performing an `overwrite operation` or has recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMovePolynomial , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMovePtp

Description

data type	Meaning
Bool	Checks whether the group is currently performing a `point-to-point transfer` or has recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMovePtp Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveOrientation

Description

data type	Meaning
Bool	Checks whether the group is currently performing an `orientation movement` or has recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveOrientation Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveExternal

Description

data type	Meaning
Bool	Checks whether the group is currently performing a `coupled movement` or has recently performed one.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveExternal , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

68.2 Example task: MCGeo.State

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample 1 : Autorun

    //Unlock the group
    If geoGroup.State.IsDisabled Then
        geoGroup.Power.Enable();
    End_if;
    When geoGroup.State.IsEnabled , continue;

    //A process may be associated with the group here, or other tasks may
move the group
    //…

    // Lock the group
    If geoGroup.State.IsRunning Then //if the group is still moving
        geoGroup.MoveHalt.Start(); //Anhalten des Gruppenbetriebs
    When geoGroup.State.IsEnabled Continue
    End if;

    geoGroup.Power.Disable()//Disable the group
    If geoGroup.State.IsDisabled , continue

End_Task;
```

69 MCGeo.State.Transitions

The **MCGeo.State.Transitions** function allows you to control state transitions in a path group for specific applications.

With the exception of the Suspend() and Resume() functions, state transitions in a group are generally triggered automatically by motion commands and the behavior of the axes. Direct control of state administration is required only in exceptional cases—such as when responding to an error. Using the **MCGeo.State.Transitions**function, as well as `Diagnostics.ClearErrors()`, state transitions can be explicitly triggered.

The **MCGeo.State** function allows you to retrieve various details about a path group, such as operating state, movement type, and so on.

Basic and additional information

In this context, please also note the following information:

– For a description of the other functions of MCGeo.State, see the section MCGeo.State [▶ 376];
– For a detailed description of the MCX core architecture, see the JetSymonline help
– For basic information, see Motion Control Application Note No. 54 (MCX Motion Control Basics).

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN) for procedures and sample projects related to working with groups.** These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information about path groups.

operating state

The operating state (`OperatingMode`) specifies the general way in which the path group operates. There are 3 operating states:

operating state	Meaning
`Inactive`	The group cannot accept commands, for example, because it has not been activated.
`Normal` (normal)	The group is usable. Traverse commands are executed in a buffered manner (see MCGeo.Buffer [▶ 256]). In other words, all movement commands are stored in a buffer and processed one after another.
`Error`	An operator error has occurred. The operating state is retained; however, no new functions or parameters can be passed.

Tab. 8: Operating states of MCX path groups

Operating status

At any given time, a group is in one, and only one, clearly defined operating state (`OperatingState`). For example, if the group is in the **"Running"**operating state, the **"Enabled"** operating state is not active, even though all group axes naturally have pulse enable.

In a given state, only a subset of all function calls is permitted. The state model for the axis is as follows:

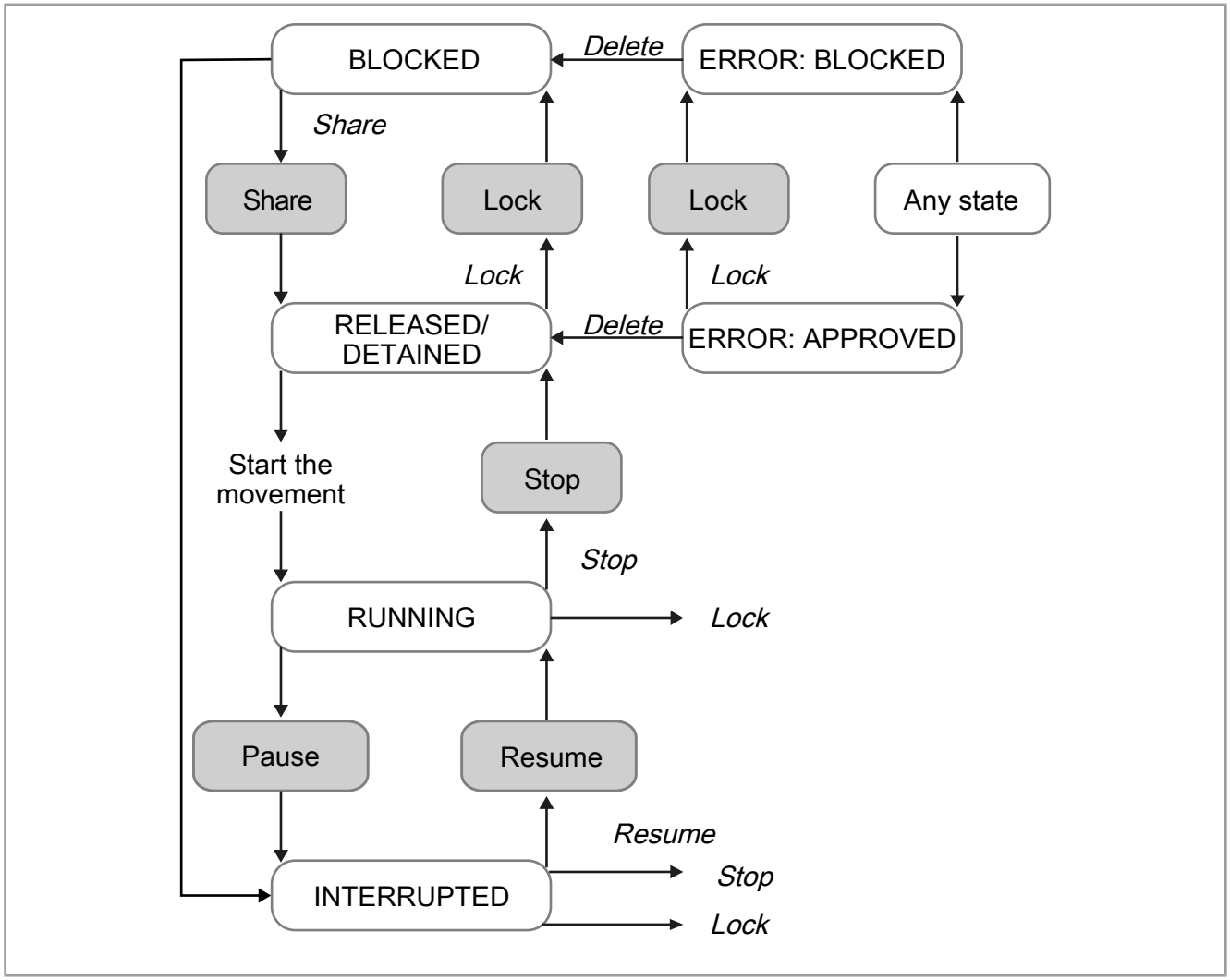


Fig. 14: State-space model for MCX groups

Operating status	Meaning
Inactive	The group cannot accept commands, for example, because it has not been activated.
Disabled	– The servo amplifier controllers for all coupled axes are open (no pulse enable). – No torque is generated.
Enabled	– The controllers for all coupled axes are closed (pulse enable). – The setpoint interpolator is at a standstill.
Running (In progress)	At least one axis is in motion; that is, the setpoint interpolator transmits motion setpoints to the servo amplifier.
Suspended	– This corresponds to the **"Enabled"** operating state, with the exception that the target position is retained. A movement that has been paused using `MCGeo.State.Transitions.Suspend()` can be resumed using the command `MCGeo.State.Transitions.Resume()`.
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control mode (error level 4).
DisabledError (Error: Locked)	An error has occurred. After the error response, control of all axes is disabled (error levels 5 and 6).

Tab. 9: Operating statuses of path groups

Type of movement

The motion type (`TypeOfMotion`) specifies the type of motion that the setpoint generator is currently generating for the path group.

Querying the transaction type is useful, for example, in a monitoring task where the process is unknown. Based on the active motion type prior to shutdown, it is possible to control the restart correctly.

Operating status	Meaning
Not a move	This transaction type remains active until the first transaction in the group is executed.
Straight (`MoveLinear`)	The path group is currently moving in a straight line, or was moving in a straight line just a moment ago.
Circular Motion 2D (`MoveCircular2D`)	The path group is currently performing, or most recently performed, a 2D circular motion.
Circular Motion 3D (`MoveCircular3D`)	The path group is currently performing, or was most recently performing, a 3D circular motion.
Spline (`MoveSpline`)	The path group is currently executing, or most recently executed, a spline motion.
Polynomial (`MovePolynomial`)	The path group is undergoing a restructuring.
Point to Point (`MovePtp`)	The path group is currently moving, or was most recently moving, from point A to point B.
Orientation (`MoveOrientation`)	The path group is currently performing, or has recently performed, a turning maneuver.
Linked movement (`MoveExternal`)	The path group is currently, or was most recently, operated using a linked movement.

69.1 Syntax for MCGeo.State.Transitions

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.State.Transitions.<Function>(optional <Parameter>);
```

69.2 Functions of MCGeo.State.Transitions

This function offers the following functions:

- Enable()
- Disable()
- Stop()
- ErrorDisable()
- ErrorStop()
- ErrorLockUserInput()
- Suspend()
- ErrorSuspend()
- Resume()

69.2.1 Enable()

The `State.Transitions.Enable()` function is identical to the `MCGeo.Power.Enable()` function. For a detailed description of the function, see the MCGeo.Power.Enable() [▶ 370]section.

Syntax

```
geoGroup.State.Transitions.Enable();
```

69.2.2 Disable()

Depending on the parameter passed (see the table below), this function performs the same action as the `MCGeo.Power.Disable()` and `MCGeo.Power.Quickstop()` commands. For more details, please refer to the descriptions for MCGeo.Power [▶ 370].

transfer parameter

Parameters	data type	Meaning
Mode	MCStateTransitionDisableModes	Mode for executing the `Disable()` function
	MCStateTransitionDisable-Modes.ImmediatePulseBlocking	Performs the same function as the `MCGeo.Power.Disable()` command or `MCGeo.Power.Disable(MCPowerDisable-Modes.Forced)`
	MCStateTransitionDisable-Modes.VelocityControlSlope	Performs the same function as the `MCGeo.Power.QuickStop()` command.

Syntax

```
geoGroup.State.Transitions.Disable(MCStateTransitionDisableModes.Immedi-
atePulseBlocking);
geoGroup.State.Transitions.Disable(MCStateTransitionDisableModes.VelocityCon
trolSlope);
```

69.2.3 Stop()

Description

Depending on the parameter passed (see table below), this function performs the same action as the MCGeo.Stop() [▶ 399]command.

The difference is that passing a function parameter is mandatory when using `MCGeo.State.Transition.Stop()` , whereas it is optional when using `MCGeo.Stop()` (see there for details).

transfer parameter

Parame-ters	data type	Meaning
Mode	MCStateTransitionStop-Modes	There are two delay values available.
	StopModes.ProgDecel-eration	The deceleration value last specified for the group is used for stop-ping.
	StopModes.MaxDeceler-ation	To bring the system to a stop, the maximum acceleration of the group is used as deceleration. This was either specified during commissioning in the motion setup under "Maximum Deceleration" or programmed using the command `geoGroup.Mechanism.Limitations.MaxAcceleration .`

Syntax

```
geoGroup.State.Transitions.Stop(MCStateTransitionStopModes.ProgDecelera-
tion);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	X		Blocked
Approved	√		Approved
Working	√	Working	Approved
Interrupted	√		Approved
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Condition Monitoring

Various pieces of information are available to help you monitor the positioning.

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis is stopped.

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator for at
least one joint axis is generating motion.
```

– The **slope status** indicates which phase of the motion cycle the setpoint generator is currently in.

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

– This function does not affect the **target window status** .

69.2.4 ErrorDisable()

This function triggers an error in the path group, resulting in the **"Error: Locked"** state. The error response can be configured (see the `ErrorDisableMode` property).

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.State.Transitions.ErrorDisable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: blocked
Approved	√	Error: blocked
Working	√	Error: blocked
Interrupted	√	Error: blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

69.2.5 ErrorStop()

This function triggers an error in the group, resulting in the **"Error: Released"** state. The error response can be configured (see the ErrorStopMode [▶ 397]property).

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.State.Transitions.ErrorStop();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: Approved
Approved	√	Error: Approved
Working	√	Error: Approved
Interrupted	√	Error: Approved
Error: Approved	-	Error: Approved

Initial state	Allowed	Final state
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

69.2.6 ErrorLockUserInput()

This function triggers an operational error in the group.

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.State.Transitions.ErrorLockUserInput();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

69.2.7 Suspend()

Use this function to interrupt a group movement that is currently in progress. Deceleration occurs either using the last programmed deceleration slope or the configured maximum deceleration. The state after stopping is not **"Released"** but **"Paused**." The `Resume()` function allows you to resume the interrupted path movement of a group and move it to its original target position. If the buffer is not yet empty, the remaining transactions are appended to it.

The slope settings (acceleration and deceleration) cannot be changed while the movements in the pre-run buffer are being executed, since the pre-calculated movements are stored in the pre-run buffer. Therefore, the value used for the programmed delay in the `Suspend()` command is the same as the one used for the path movements in the pre-run buffer. The value can only be changed after the transactions in the pre-processing buffer have been completed.

transfer parameter

Parameters	data type	Meaning
Mode	MCStateTransitionStopModes	Mode for the deceleration slope of the `Sus-pend()` function
	MCStateTransitionStopModes.Pro-grammedDeceleration	Uses the delay last programmed for the group.
	MCStateTransitionStopModes.MaxDe-celeration	Uses the maximum delay configured for the group.

Syntax

```
geoGroup.State.Transitions.Suspend(MCStateTransitionStopModes.ProgDecelera-
tion);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	X	Approved
Working	√	Interrupted
Interrupted	–	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

69.2.8 ErrorSuspend()

This function triggers an error in the path group, resulting in the **"Error: Released"** state. The path movement is stopped after the programmed delay. The pre-run buffer is not cleared, so the operating state remains **"Interrupted"** after the error is cleared, and motion can be resumed using Resume() [▶ 395] .

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.State.Transitions.ErrorSuspend();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: Approved
Approved	√	Error: Approved
Working	√	Error: Approved

Initial state	Allowed	Final state
Interrupted	√	Error: Approved
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

69.2.9 Resume()

This function resumes positioning that was interrupted by the Suspend() [▶ 393] function. The system moves to the originally specified target position, and any additional movements stored in the pre-run buffer are appended. The velocity and acceleration values are also taken from the original path movement.

The `Resume()` function only takes effect if the group is in the **"Paused"** state.

In buffered mode, the output buffer is not cleared.

transfer parameter

The function has no transfer parameters.

Syntax

```
geoGroup.State.Transitions.Resume();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	X	Approved
Working	–	Working
Interrupted	√	Working
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

69.3 Properties of MCGeo.State.Transitions

This function offers the following properties:

– ErrorDisableMode

– ErrorStopMode

– ErrorLevel

69.3.1 ErrorDisableMode

Description

data type	Meaning
MCStateTransitionDisableModes	Error response used when the `ErrorDisable()` function is called.
MCStateTransitionDisableModes.Velocity-ControlSlope	Stops the group using the emergency brake. Once a standstill is reached, the servo amplifier is disabled.
MCStateTransitionDisableModes.Immedi-atePulseBlocking	Immediately disables the servo amplifier for torque generation (pulse inhibit).

Syntax

```
geoGroup.State.Transitions.ErrorDisableMode := MCStateTransitionDisable-
Modes.ImmediatePulseBlocking;
Error response := geoGroup.State.Transitions.ErrorDisableMode;
```

state administration

The property can be read and written in any state.

When the property is read, the currently valid error response for the command is returned.

69.3.2 ErrorStopMode

Description

data type	Meaning
MCStateTransitionErrorStopModes	Error response used when the `ErrorStop()` function is called.
MCStateTransitionErrorStop-Modes.ProgrammedDeceleration	Pauses the group for the programmed delay.
MCStateTransitionErrorStop-Modes.MaxDeceleration	Pauses the group for the maximum delay.
MCStateTransitionErrorStop-Modes.AtBlockEnd	Holds the group in place at the end of the current movement. Processing of the pre-processing buffer is stopped.

Syntax

```
geoGroup.State.Transitions.ErrorStopMode := MCStateTransitionStop-
Modes.MaxDeceleration;
Error response := geoGroup.State.Transitions.ErrorStopMode;
```

state administration

The property can be read and written in any state.

69.3.3 ErrorLevel

Description

Use this property to set the error level that the path group adopts in the event of a drive error. By default, this is error level 5 (stop with emergency stop slope and operating status **: error—locked**).

data type	Meaning
MCStateTransitionErrorLevels	Error level of the path group in the event of a drive error. For detailed information on the differences between error response levels, see the " *Error Management"*section in the JetSym online help.
MCStateTransitionErrorLevels.Message	The group responds with error level 1: A notification/alert is generated.
MCStateTransitionErrorLevels.Channel-Stop	The group responds with error level 2: The command channel is being blocked.
MCStateTransitionErrorLevels.Channel-StopClearBuffer	The group responds with error level 3: The command channel is locked, and the preemptive buffer is cleared.
MCStateTransitionErrorLevels.Stop	The group responds with error level 4: A system stop is being performed.
MCStateTransitionErrorLevels.Quickstop (Default)	The group responds with error level 5: An emergency stop is initiated.
MCStateTransitionErrorLevels.PulseBlocking	The group responds with error level 6: The controller is disabled (pulse disable).

Syntax

```
geoGroup.State.Transitions.ErrorLevel := MCStateTransitionErrorLevels.Puls-
Blocking;
ReactionOnDriveError := geoGroup.State.Transitions.ErrorLevel;
```

state administration

The property can be read and written in any state.

70 MCGeo.Stop

The **MCGeo.Stop** function allows you to stop a moving path group using a defined slope.

Unlike the `MoveHalt.Start()` function, `Stop` clears the feed buffer, meaning that all motion blocks still in the buffer are deleted.

This function has no additional functions or properties.

ⓘ Info

Further information

Please refer to the **Motion Control Application Notes (APN)**. These are available for download on the Bucher Automation website.

70.1 Syntax for MCGeo.Stop

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Stop(optional <Parameter 1>)
```

70.2 Features of MCGeo.Stop

`MCTGeo.Stop` is a method of the group object.

70.2.1 Stop()

transfer parameter

Please note the following guidelines:

– This function executes the stop command immediately. The parameter passed (see table) specifies the maximum delay for this.

– If no parameter is passed, the last `StopMode` passed is used.

– The group stops at every station.

Parameters	data type	Meaning
StopModes	Enum StopModes (optional)	There are two delay values available.
	StopModes.ProgDeceleration	The deceleration value last specified for the group is used for stopping (default).
	StopModes.MaxDeceleration	To stop, the maximum acceleration of the axis is used as deceleration. This was either specified during commissioning in the motion setup under "Maximum Deceleration" or programmed using the command `geoGroup.Mechanism.Limitations.MaxAcceleration` .

Syntax

```
geoGroup.Stop(StopModes.ProgDeceleration);
geoGroup.Stop(StopModes.MaxDeceleration);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	X		Blocked
Approved	√		Approved
Working	√	Working	Approved
Interrupted	√		Approved
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning.

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis is stopped.

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator for at
least one joint axis is generating motion.
```

– The **slope status** indicates which phase of the motion cycle the setpoint generator is currently in.

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

– This function does not affect the **target window status** .

71 MCManager (MCMgr)

The **MCMgr** function allows you to read information about the axis and group objects defined in the project's hardware tree. This information can be used, for example, in the STX program for loop commands that apply to all or some of the axis objects.

71.1 Syntax for MCMgr

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
MCMgr.<Function or Access to Value>(optional <Parameter 1, Parameter 2,
Parameter n>);
```

71.2 Features of MCMgr

This function offers the following functions:

– CheckVersions()
– GetDataSource()
– WriteLogToFile()
– GetAxisByID()
– GetAxisByName()
– GetTechnoByID()
– GetTechnoByName()
– GetGeoByID()
– GetGeoByName()

71.2.1 CheckVersions()

This feature allows you to compare the version of the MCX kernel you are using with the minimum MCX kernel version required for the Motion API version you are using. This allows you to verify whether the controller's operating system software (including the MCX kernel) is compatible with the functions of the Motion API version.

If the MCX kernel version is lower than the minimum required version, `false` is returned. In this case, the JetSym trace window displays the minimum required OS version for the controller.

transfer parameter

This function has no transfer parameters.

Syntax

```
if Not MCMgr.CheckVersions() then
    Taskbreak tInit;
end_if;
```

state administration

The function can be read in any state.

71.2.2 GetDataSource()

transfer parameter

Parameters	data type	Meaning
ShadowAxis	Pointer to MCAxis	Reads the axis object that serves as the data axis for the follow-up axis passed to the function.

Syntax

```
DataAxis := MCMgr.GetDataSource(axShadow);
```

state administration

The function can be read in any state.

71.2.3 WriteLogToFile()

This function writes the contents of the MCX logger's buffer to a text file named *MCLogger.dat* , which is saved in the JetControl file system. This can be used to save the MCX actions during a specific program sequence (e.g., in the event of an error) so that they can be analyzed later. The file contains the logger messages encoded as numbers. To analyze the data, you must transfer the file from the controller to the PC and decode it into text.

Transfer the data file to the PC and decode it

1. Use an FTP tool or JetSym to transfer the *MCLogger.dat* file from the controller to the computer.
2. In JetSym open the Hardware > MC-Global tab.
3. Select motion setup > Status.
4. Make sure that the setup is disabled (offline mode).
5. Click "Open Log File" and select the location of the *MCLogger.dat* file.
   ⇨ Another dialog box will open, allowing you to save the decoded text file to your computer.
6. Follow the instructions.

⇨ The logger messages are displayed as text in the status section.

transfer parameter

Parameters	data type	Meaning
MaxLength	Integer (optional)	Defines the length of the most recently recorded data in the buffer [in bytes] that is to be transferred to the file. If no parameter is provided or `0` is passed, the entire buffer that has been recorded so far is written to the file. The length depends on the controller type and can be up to 40 KB. The transfer is performed in the background and may take some time.

Syntax

```
MCMgr.WriteLogToFile(255);
```

state administration

The function can be read in any state.

71.2.4 GetAxisByID()

The axis's ObjectID is assigned by JetSym and must be unique within a project. In JetSym it is displayed as the axis ID in the tooltip of the Hardware tree and on the Axis Parameters page of the Motion Setup.

transfer parameter

Parameters	data type	Meaning
ObjectID	Integer	Reads the axis object (return value of type `Pointer to MCAxis`) associated with the passed ObjectID.

Syntax

```
MCAxis := MCMgr.GetAxisByID(8);
```

state administration

The function can be read in any state.

71.2.5 GetAxisByName()

The axis name must be unique within a project. It is assigned by JetSym when the axis is added to the Hardware tree, but it can be customized.

transfer parameter

Parameters	data type	Meaning
Name	String	Reads the axis object (return value of type `Pointer to MCAxis`) associated with the specified name string.

Syntax

```
MCAxis := MCMgr.GetAxisByName('AX1');
```

state administration

The function can be read in any state.

71.2.6 GetTechnoByID()

The ObjectID for the technology group is assigned by JetSym and must be unique within a project. In JetSym it is displayed as a TechnoID in the tooltip of the hardware tree and on the Properties page of the motion setup.

transfer parameter

Parameters	data type	Meaning
ObjectID	Integer	Reads the group object (return value of type `Pointer to MCTechno`) associated with the passed ObjectID.

Syntax

```
MCTechGroup := MCMgr.GetTechnoByID(5);
```

state administration

The function can be read in any state.

71.2.7 GetTechnoByName()

The consortium name must be unique within a project. It is assigned by JetSym when the group is added to the hardware tree, but it can be customized.

transfer parameter

Parameters	data type	Meaning
Name	String	Reads the group object (return value of type `Pointer to MCTechno`) associated with the specified name string.

Syntax

```
MCTechno := MCMgr.GetTechnoByName('AV1');
```

state administration

The function can be read in any state.

71.2.8 GetGeoByID()

The ObjectID for the path group is assigned by JetSym and must be unique within a project. In JetSym it is displayed as a GeoID in the tooltip of the hardware tree and on the Properties page of the motion setup.

transfer parameter

Parameters	data type	Meaning
ObjectID	Integer	Reads the group object of a path group (return value of type `Pointer to MCGeo`) associated with the passed ObjectID.

Syntax

```
MCGeoVerbund := MCMgr.GetGeoByID(2);
```

state administration

The function can be read in any state.

71.2.9 GetTechnoByName()

The consortium name must be unique within a project. It is assigned by JetSym when the group is added to the hardware tree, but it can be customized.

transfer parameter

Parameters	data type	Meaning
Name	String	Reads the group object of a path group (return value of type `Pointer to MCTechno`) associated with the provided name string.

Syntax

```
MCGeo := MCMgr.GetGeoByName('Bahn1');
```

state administration

The function can be read in any state.

71.3 Properties of MCMgr

This function offers the following properties:

- AxisCount
- AxisList
- TechnoCount
- TechnoList
- GeoCount
- GeoList
- Number of Faulty Objects
- FaultyObject
- Name
- Version
- IsInitialized

71.3.1 AxisCount

Description

data type	Meaning
Integer	Reads the number of axis objects in the project. This can be particularly helpful for loop commands, where all axes in a project are supposed to perform the same function one after another.

Syntax

```
//This sequence locks all MCX axes defined in the project
for nAxisIndex := 0 to mcMgr.AxisCount - 1, incrementing by 1, do
    mcMgr.AxisList[nAxisIndex].Power.Disable();
end_for;
```

state administration

The property can be read in any state.

71.3.2 AxisList[]

Description

data type	Meaning
Array of pointers to MCAxis	Accesses the project's MCX axes using an index. This can be particularly helpful for loop commands, where all axes in a project are supposed to perform the same function one after another.

Syntax

```
//This sequence locks all MCX axes defined in the project
for nAxisIndex := 0 to mcMgr.AxisCount - 1, incrementing by 1, do
    mcMgr.AxisList[nAxisIndex].Power.Disable();
end_for;
```

state administration

The property can be read in any state.

71.3.3 TechnoCount

Description

data type	Meaning
Integer	Reads the number of technology-related objects in the project. This can be particularly helpful for loop statements, where all the technology groups in a project are supposed to perform the same function one after another.

Syntax

```
//This sequence activates all technology groups defined in the project
for nTechnoIndex := 0 to mcMgr.TechnoCount - 1, incrementing by 1, do
    mcMgr.TechnoList[nTechnoIndex].Activate();
end_for;
```

state administration

The property can be read in any state.

71.3.4 TechnoList[]

Description

data type	Meaning
Array of pointers to MCTechno	Access the project's technology groups using an index. This can be particularly helpful for loop statements, where all the technology groups in a project are supposed to perform the same function one after another.

Syntax

```
//This sequence activates all technology groups defined in the project
for nTechnoIndex := 0 to mcMgr.TechnoCount - 1, incrementing by 1, do
    mcMgr.TechnoList[nTechnoIndex].Activate();
end_for;
```

state administration

The property can be read in any state.

71.3.5 GeoCount

Description

data type	Meaning
Integer	Reads the number of path group objects in the project. This can be particularly helpful for loop commands, where all path groups in a project are supposed to perform the same function one after another.

Syntax

```
//This sequence activates all path groups defined in the project
for nGeoIndex := 0 to mcMgr.GeoCount - 1, incrementing by 1, do
    mcMgr.GeoList[nGeoIndex].Activate();
end_for;
```

state administration

The property can be read in any state.

71.3.6 GeoList[]

Description

data type	Meaning
Array of pointers to MCTechno	Access the project's path groups using an index. This can be particularly helpful for loop commands, where all path groups in a project are supposed to perform the same function one after another.

Syntax

```
//This sequence activates all path groups defined in the project
for nGeoIndex := 0 to mcMgr.GeoCount - 1, incrementing by 1, do
    mcMgr.GeoList[nGeoIndex].Activate();
end_for;
```

state administration

The property can be read in any state.

71.3.7 Number of Faulty Objects

Description

data type	Meaning
Integer	Reads the number of MCX objects in the project that are in an error state. Reports are not taken into account in this process. For an example of use, see FaultyObjectList[] [▶ 407]property.

state administration

The property can be read in any state.

71.3.8 FaultyObjectList[]

Description

data type	Meaning
Array of pointers to MCTechno	List the MCX objects in the project that are reporting errors. Reports are not taken into account in this process. Using this list, you can, for example, retrieve the error number from the identified MCX object.

Syntax

```
//This sequence checks for errors in the axes, techno objects, or path
groups, and identifies the axis and group objects that are showing errors.
The number of the last error is stored in a variable.

var
    AxErrorNo : int;
    TecErrorNo : int;
    GeoErrorNo : int;
end_var;
```

```
Task tErrorRead

Var
    n, i : int;
    mcObj : IMCObject; //common interface for MCX objects
    mcAx : pointer to MCAxis;
    mcTec: pointer to MCTechno;
    mcGe : pointer to MCGeo;
End_Var;


    AxErrorNO := 0;
    TecErrorNo := 0;
    GeoErrorNo := 0;

    n := mcMgr.FaultyObjectCount;

    if n > 0 then
        for i := 1 to n do
            mcObj := mcMgr.FaultyObject[i-1];

         Trace('Invalid object: ' + mcObj.Info.Name + '$n');

            // Check whether the faulty object is an axis
            if mcObj.Info.ClassID == MCClassIDs.Axis then
                // Get the axis with the specified object ID
                mcAx := mcMgr.GetAxisByID(mcObj.Info.ObjectID);
                // Read error code
                AxErrorNo := mcAx.Diagnostics.GetErrorCode();
            end_if

            // Check whether the faulty object is a techno composite group
            if mcObj.Info.ClassID == MCClassIDs.Techno then
                // Retrieve the Techno object with the specified object ID
                mcTec := mcMgr.GetTechnoByID(mcObj.Info.ObjectID);
                // Read error code
                TecErrorNo := mcTec.Diagnostics.Software.GetErrorCode();
            end_if;

            // Check whether the faulty object is a path group
            if mcObj.Info.ClassID == MCClassIDs.Geo then
                // Retrieve the geo-object with the specified object ID
                mcGe := mcMgr.GetGeoByID(mcObj.Info.ObjectID);
                // Read error code
                GeoErrorNo := mcGe.Diagnostics.Software.GetErrorCode()
            end_if;

        end_for;
    end_if;

End_Task
```

state administration

The property can be read in any state.

71.3.9 Name

Description

data type	Meaning
String	Reads the type of MCX controller being used. The return value is a string (e.g., *JC-965EXT*).

state administration

The property can be read in any state.

71.3.10 Version

Description

data type	Meaning
Integer	Reads the version of the MCX kernel on the controller. The display is in IP format.

state administration

The property can be read in any state.

71.3.11 IsInitialized

Description

data type	Meaning	
Integer	Checks whether the MCX kernel was fully initialized with all axes during bootup and is ready for operation. `True`: The MCX kernel has been fully initialized during boot and is ready for operation `False`: Not all MCX axes were detected on the EtherCAT, or their initialization failed. In the MCX Logger, this is indicated by the following lines:	
	11:01:42:112	`Setup: Synchronous drives`
	11:01:42:112	`Setup: MC V. 1.2.114.2 successfully initialized and started`

Syntax

```
When mcMgr.IsInitialized , continue;
```

state administration

The property can be read in any state.

72 MCTechno.Activate, MCTechno.Deactivate

You can use the **MCTechno.Activate** and **MCTechno.Deactivate** functions to activate and deactivate a technology group. Changes to a technology group are permitted in all operating states of the group, except when **it is inactive** .

Once a technology group has been activated, all member axes are set to the **inactive** operating state. You can now only access these axes through the path group to which they are connected.

A technology group can only be enabled or disabled if all member axes in that group are either **enabled** or **disabled** , and all member axes have the same operating state. Otherwise, an error message will be displayed.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

72.1 Syntax for MCTechno.Activate, MCTechno.Deactivate

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Activate();
<Axis group name>.Deactivate();
```

72.2 Functions of MCTechno.Activate and MCTechno.Deactivate

This function offers the following functions:

– Activate()

– Deactivate()

72.2.1 Activate()

This function activates a technology group provided that all member axes are in either the **Locked** or **Unlocked** state.

transfer parameter

The function has no transfer parameters.

Syntax

```
techGroup.Activate();
```

state administration

The following table applies to the reference and follow axes when calling the function:

Initial state	Allowed	Final state
Inactive	√	Locked/Unlocked
Blocked	–	Blocked
Approved	–	Approved

Initial state	Allowed	Final state
Working	–	Working
Interrupted	–	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Check whether the new state has been reached in the program before issuing any further commands to the axes.

```
If Not techGroup.State.IsInactive , Continue;
```

Alternatively, you can also check the **IsEnabled** or **IsDisabled** status of the member axes—depending on the status of the group before the command was executed.

72.2.2 Deactivate()

Use this feature to disable a technology group. After this command, the group is in an **inactive**state. The member axes of the group can once again be controlled independently. They are taking over the operational management of the technology group.

transfer parameter

The function has no transfer parameters.

Syntax

```
techGroup.Deactivate();
```

state administration

The following table applies to the reference and secondary axes when calling the function:

Initial state	Allowed	Final state
Inactive	–	Inactive
Blocked	√	Inactive
Approved	√	Inactive
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Check whether the new state has been reached in the program before issuing any further commands to the axes.

```
When techGroup.State.IsInactive , continue;
```

Alternatively, you can also check the **IsEnabled** or **IsDisabled** status of the member axes—depending on the status of the group before the command was executed.

73 MCTechno.Coupling

The **MCTechno.Coupling** function allows you to manage the coupling function of the technology group.

A detailed description of the functions associated with the `Cam` property is provided in a separate section (MCTechno.Coupling.Cam [▶ 426]).

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

73.1 Syntax for MCTechno.Coupling

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Coupling.<Function or Access to Value>(optional <Parameter 1, Parameter 2, Parameter n>);
```

73.2 Features of MCTechno.Coupling

This function offers the following functions:

- Couple()
- Decouple
- GetCoupleMasterPosition()
- GetCoupleFollowerPosition()
- GetCoupleMasterDirection()
- GetCoupleMode()
- GetDecoupleDeceleration()
- GetDecoupleMasterPosition()
- GetDecoupleMode()
- IsCoupled()
- IsCoupling()
- IsDecoupled()
- IsDecoupling()
- Filter.GetCoefficient()
- Filter.SetCoefficient()
- Gearing.Configure()
- Gearing.GetFactor()
- Gearing.GetDivisor()
- Gearing.GetActivationMode()

73.2.1 Couple()

This function allows you to initiate the coupling of a secondary axis within a technology group. This process requires that the group has been activated. Depending on the alignment mode, coupling is either performed directly, or a process is initiated that performs the coupling based on various parameters of the master and slave axes. During this process, the coupling state is *IsCoupling*. At the end of the process, the coupling status is *IsCoupled*.

A message is displayed when the function is executed on the reference axis.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A follow-up object that must be a member of the technology group and is to undergo coupling.
Couple Mode	MCTechnoCoupleModes (optional)	The mode in which the coupling is to be performed. Familiarize yourself with the specific procedures used in the four coupling modes: – For an electric gear: Application Note 050 – For a cam disc: Application Note 056
	MCTechnoCoupleModes Immediate	The coupling is performed immediately and takes effect right away.
	MCTEechnoCouple-Modes ImmediateSnap	The coupling is performed immediately and takes effect right away. – With an **electric gear**, the behavior is identical to that of the *"Immediate"* engagement mode. – In the case of a **cam disc**, the cam disc is moved so that the setpoint matches the setpoint before coupling.
	MCTechnoCoupleModes Fast	The coupling is performed: – with acceleration via a coupling polynomial to the coupling position (see MasterPosition and Follwower Position); – if necessary, with an increase in the coupling speed;
	MCTechnoCoupleModes Wait	The coupling is performed – accelerating to the coupling position (see MasterPosition and FollowerPosition); – if necessary, by waiting for a module reset to prevent the coupling speed from exceeding the limit;
Master's Degree	Double (optional)	For the *Fast* and *Wait* synchronization modes: The position in the master slot where the coupling should be completed. The value must be greater than `0`.
MasterDirection	Directions (optional)	For the *Fast* and *Wait* synchronization modes: Direction of movement of the reference axis. Do not perform coupling when the direction of movement is opposite.
	Directions.Positive	The trend is moving in a positive direction.
	Directions.Negative	The reference axis is moving in the negative direction.
Follower Position	Double (optional)	For the *Fast* and *Wait* synchronization modes: The position on the axis where the coupling should be completed.

Syntax

```
tecGroup.Coupling.Couple(Follower1, MCTechnoCoupleModes.Fast,1000.0, Direc-
tions.Positive,500.0);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Error: blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

73.2.2 Decouple()

This function allows you to decouple a secondary axis in a technology group. This process requires that the group has been activated. Depending on the decoupling mode, decoupling is either performed immediately, or a process is initiated that performs the decoupling based on various parameters of the master and slave axes. During this process, the coupling state is *IsDecoupling*. At the end of the process, the coupling status is *IsDecoupled*.

A message is displayed when the function is executed on the reference axis.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A child object that must be a member of the technology group and is to be decoupled.

Parameters	data type	Meaning
Decouple Mode	MCTechnoDecouple-Modes (optional)	The mode in which the disengagement is to be performed. Familiarize yourself with the specific procedures used in the three decoupling modes: – For an electric gear: Application Note 050 – For a cam disc: Application Note 056
	MCTechnoDecouple-Modes Immediate_Stop	The disengagement is performed immediately. The follow-up axis is stopped with the deceleration specified in the `Decel-eration` parameter.
	MCTEechnoDecouple-Modes_KeepVelocity	The disengagement is performed immediately. The follow axis continues to move at the speed it had when it was decoupled (similar to the `MoveVelocity()` command). If the lead axis is in the stop slope, the following axis does not maintain its speed in *Immediate_KeepVelocity* mode, but is stopped with a specified deceleration, as in *Immediate_Stop* mode.
	MCTechnoDecouple-Modes.AtMasterPosi-tion_Stop	The slave axis is decoupled as soon as the master axis reaches the position value specified in the `MasterPosition` parameter. It is then brought to a stop with the deceleration de-fined in the `Deceleration` parameter.
Master's Degree	Double (optional)	Disengagement position in *AtMasterPosition_Stop* disengage-ment mode.
Deceleration	Double (optional)	Delay for stopping the following axis after disengagement.

Syntax

```
tecGroup.Coupling.Decouple(Follower1, MCTechnoDecoupleModes.Immediate_Stop);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Error: blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

73.2.3 GetCoupleMasterPosition()

This function reads the reference axis position—last specified using the Couple function—at which the coupling process is to be completed. (Active only in *Fast* and *Wait* modes)

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the position of the lead object in the technology group at which the coupling should be completed. (Active only in *Fast* and *Wait* modes)

Syntax

```
LeitAchs_KoppelPos := tecGroup.Coupling.GetCoupleMasterPosi-
tion(axLeitachse);
```

state administration

The function can be read in any state.

73.2.4 GetCoupleFollowerPosition()

This function reads the last secondary axis position specified using the Couple function, at which the coupling process is to be completed. (Active only in *Fast* and *Wait* modes)

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the position of the subsequent axis object in the technology group where the coupling is to end. (Active only in *Fast* and *Wait* modes)

Syntax

```
FolgeAchs_KoppelPos := tecGroup.Coupling.GetCoupleFollowerPosition(Fol-
lower1);
```

state administration

The function can be read in any state.

73.2.5 GetCoupleMasterDirection()

This function reads the direction of the main axis movement that was last specified using the Couple function. (Active only in *Fast* and *Wait* modes)

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the direction of movement of the leading axis object in the technology group whose direction of movement is to be read. (Active only in *Fast* and *Wait* modes)

Syntax

```
LeitAchs_Richtung := tecGroup.Coupling.GetCoupleMasterDirec-
tion(axLeitachse);
```

state administration

The function can be read in any state.

73.2.6 GetDecoupleDeceleration()

This function reads the delay value used for decoupling in the *Immediate_Stop* and *AtMasterPosition_Stop* decoupling modes.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the delay value for the follow-up axis object of the technology group that is to be read for decoupling

Syntax

```
DecouplingDeceleration := tecGroup.Coupling.GetDecoupleDeceleration(Follower1);
```

state administration

The function can be read in any state.

73.2.7 GetDecoupleMasterPosition()

This function retrieves the decoupling position most recently passed to the Decouple function. (Only in *AtMasterPosition_Stop* decoupling mode.)

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the position of the follower axis object in the technology group whose master axis decoupling position is to be read. (Only for the *AtMasterPosition_Stop* decoupling module)

Syntax

```
LeitAchs_DecouplingPosition := tecGroup.Coupling.GetDecoupleMasterPosition(Follwer1);
```

state administration

The function can be read in any state.

73.2.8 GetDecoupleMode()

This function allows you to read the mode used for decoupling.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the decoupling mode applied to the subsequent axis object in the technology group.

Syntax

```
DecouplingMode := tecGroup.Coupling.GetDecoupleMode(Follower1);
```

state administration

The function can be read in any state.

73.2.9 IsCoupling()

This function corresponds to the `IsCoupling` property of the follow-up axis (MCAxis.State.Techno.IsCoupling [▶ 214]).

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling status of a slave axis. `IsCoupling` is `true` if the axis has received the coupling command in the *"almost"* and *"wait"* coupling modes but has not yet reached the coupling point.

Syntax

```
When tecGroup.Coupling.IsCoupling(Follower1) , Continue;
```

state administration

The function can be read in any state.

73.2.10 IsCoupled()

This function corresponds to the `IsCoupled` property of the follow-up axis (MCAxis.State.Techno.IsCoupled [▶ 214]).

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling status of a slave axis. `IsCoupled` is `true`if the axis is coupled.

Syntax

```
When tecGroup.Coupling.IsCoupled(Follower1) , Continue;
```

state administration

The function can be read in any state.

73.2.11 IsDecoupling()

This function corresponds to the `IsDecoupling` property of the follow-up axis (MCAxis.State.Techno.IsDecoupling [▶ 213]).

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling status of a slave axis. `IsDecoupling` is `true` if the axis receives the decoupling command while in the *AtMasterPosition_Stop* decoupling mode but has not yet reached the coupling point.

Syntax

```
When tecGroup.Coupling.IsDecoupling(Follower1) , Continue;
```

state administration

The function can be read in any state.

73.2.12 IsDecoupled()

This function corresponds to the `IsDecoupled` property of the slave axis (MCAxis.State.Techno.IsDecoupled [▶ 213]).

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling status of a slave axis. `IsDecoupled` is `true` if the axis is decoupled.

Syntax

```
When tecGroup.Coupling.IsDecoupled(Follower1) , Continue;
```

state administration

The function can be read in any state.

73.2.13 Filter.GetCoefficient(), Filter.SetCoefficient()

`Get/SetCoefficient()` are subfunctions of the `Filter` property. These functions allow you to read or change the value for item filtering in a technology group. The parameters that can be specified are an axis of the technology group and a filter coefficient.

Please follow these rules:

– When a **reference axis** is selected, actual value filtering is performed. This is only useful if at least one slave axis is connected via actual value coupling.

– When a **follow-up axis** is selected, a filter is activated in the setpoint path.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A composite axis whose filter settings are to be changed or read.
Coefficient	Double	– Filter coefficient; the value is specified between `0` and `100` . The fact is: `0`: no filtering (default), `100`: maximum filtration efficiency;

Syntax

```
tecGroup.Coupling.Filter.SetCoefficient(Technoaxis,22.4);
FilterCoeff := tecGroup.Coupling.Filter.GetCoefficient(Technoaxis);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: blocked
Approved	√	Approved

Initial state	Allowed	Final state
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

73.2.14 Gearing.Configure()

`Configure()` is a subfunction of the `Gearing` property. This function allows you to configure the coupling parameters for a secondary axis that is configured as an electric gear.

Use the `Factor` and `divisor` parameters to set the gear ratio between the positions of the follower axis and the master axis.

Please follow these rules:

– A secondary shaft configured as **an electric gear** is directly coupled when the technology group is activated.

  If this is not appropriate for the application, the `MCTechnoGearingActivationMode` parameter can be used to disable automatic coupling when the group is activated. This requires that the group has been activated. After that, automatic coupling will remain inactive until it is reactivated.

– If the command is applied to a secondary axis that is configured as **a cam disc**, no function is executed and no error message is displayed.

NOTICE



Risk of re-parameterization of the lead-axis function

Applying the command to a reference axis may result in the reference axis function being reconfigured.

► Be sure to apply the command only to secondary axes.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A follow-up object that must belong to the technology group and whose coupling parameters are to be read or modified.
Factor	Double (optional)	Gear ratio.
divisor	Double (optional)	Gear ratio divisor: Must be greater than 0.
Activation Mode	MCTechnoGearingActivationModes (optional)	Status of the coupling upon activation of the technology group.
	MCTechnoGearingActivationModes.Coupled [Default]	Paired upon activation.
	MCTechnoGearingActivationModes.Decoupled	Decoupled when activated.

Syntax

```
tecGroup.Coupling.Gearing.Configure(Follower1, 1.0, 2.0, MCTechnoGearingAc-
tivationModes.Decoupled);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

73.2.15 Gearing.GetFactor()

`GetFactor()` is a subfunction of the `Gearing` property.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the gear ratio for a secondary shaft in a technology group configured as an electric gear. If the command is applied to a master axis or slave axis that is configured as a cam disc, a meaningless value is returned and no error message is displayed

Syntax

```
GearRatio := tecGroup.Coupling.Gearing.GetFactor(Follower1);
```

state administration

The function can be read in any state.

73.2.16 Gearing.GetDivisor()

`GetDivisor()` is a subfunction of the `Gearing` property. When applied to a master axis or to a slave axis configured as a cam disc, this command returns a meaningless value and does not display an error message.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the gear ratio divisor for a secondary shaft in a technology group configured as an electric gear.

Syntax

```
GearRatioDivisor := tecGroup.Coupling.Gearing.GetDivisor(Follower1);
```

state administration

The function can be read in any state.

73.2.17 Gearing.GetActivationMode()

`GetActivationMode()` is a subfunction of the `Gearing` property. Applying the command to a master axis or a slave axis that is configured as a cam disc returns a meaningless value and does not display an error message.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling status when the technology group is acti-vated for a trailing axis that is configured as an electric gear.

Syntax

```
GearActivation := tecGroup.Coupling.Gearing.GetActivationMode(Follower1);
```

state administration

The function can be read in any state.

73.3 Properties of MCTechno.Coupling

This function offers the following properties:

– Cam (see MCTechno.Coupling.Cam [▶ 426])
– ExternalMaster.Offset
– ExternalMaster.Smoothing
– Filter (see the description of the subfunctions under Features of MCTechno.Coupling [▶ 413])
– Gearing (see the description of the subfunctions under Features of MCTechno.Coupling [▶ 413])

73.3.1 ExternalMaster.Offset

Description

This property is only relevant if the reference axis of a technology group is an external axis.

An external axis is characterized by the fact that it cannot be controlled by the MCX kernel's setpoint generator, since it simply represents the position of a position sensor. An offset can be applied to this position using this property before it is used as a reference axis position in the technology group.

This command is ignored if the reference axis of the technology group is not an external axis.

data type	Meaning
Double	Reads or writes a position offset for the reference axis position from an external axis.

Syntax

```
ExternalPosOffset := tecGroup.Coupling.ExternalMaster.offset;
tecGroup.Coupling.ExternalMaster.Offset := 536.2;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

73.3.2 ExternalMaster.Smoothing

Description

Use the ExternalMaster.Smoothing property to read or set the smoothing state of the lead-screw position values for the external axis. When this property is enabled, the position of the external axis is smoothed based on the last four input values to compensate for any fluctuations. This feature can be used in addition to actual value smoothing.

This property is only relevant if the reference axis of a technology group is an external axis.

This command is ignored if the reference axis of the technology group is not an external axis.

data type	Meaning
MCTechnoSmoothingModes	Reads or sets the state of smoothing for the Leitachs position values for the external axis.
MCTechnoSmoothingModes.Disable	No smoothing is performed
MCTechnoSmoothingModes.Standard	The system smooths the last 4 input values to compensate for any fluctuations. This feature can be used in addition to actual value smoothing.

Syntax

```
External_Smoothing := tecGroup.Coupling.ExternalMasterSmoothing;
tecGroup.Coupling.ExternalMaster.Smoothing :=
MCTechnoSmoothingModes.Standard;
```

state administration

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

74 MCTechno.Coupling.Cam

The **MCTechno.Coupling.Cam** function allows you to manage the creation and configuration of cam discs for a follow-up axis within a technology group.

For a detailed description of the remaining features and properties of this function, please refer to the corresponding section MCTechno.Coupling [▶ 413].

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

APN 52 and APN 56 – Cam Discs

Please pay particular attention to the detailed information on the use of cam discs provided in **Application Notes 052 and 056**.

74.1 Syntax for MCTechno.Coupling.Cam

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Coupling.Cam.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

74.2 Features of MCTechno.Coupling.Cam

This function offers the following functions:

- Create()
- DefineSegment()
- DefineSegmentAuto()
- Activate()
- ReadSegment()
- Evaluate()
- AnalyzeSegment()
- AnalyzeDerivatives()

74.2.1 Create()

This function allows you to provide a cam disc. The cam disc is assigned an identifier, and memory space is allocated for it in the control system. These steps are necessary to define the segments of a cam disc, activate the cam disc, and link it.

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A follow-up object that must be a member of the technology group and for which a cam disc is to be created.
CamId	Integer	Specify a unique number to identify this cam disc.
SegmentCount	Integer	Defines the number of segments in this cam disc.

Syntax

```
tecGroup.Coupling.Cam.Create(Z_Axis,1,3);
```

state administration

As long as the active technology group is in operation, the function can be accessed in any state.

74.2.2 DefineSegment()

NOTICE



Risk of uncontrolled axis movement

Changes to the segment definitions take effect immediately.

- ► Avoid changing the segment definition while a cam disc is currently active.
- ► If it cannot be avoided, proceed with the utmost caution.

This function allows you to define, relative to the reference axis, the motion profile that should apply to a specific section of the cam disc. You can only define a segment if the cam disc has been created beforehand.

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated.

Difference between DefineSegment() and DefineSegmentAuto()

`DefineSegment()`: The polynomial segments are defined using the polynomial coefficients.

`DefineSegmentAuto()`: The polynomial segments are defined using initial and final conditions.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A follow-up object that must be a member of the technology group and for which a cam segment is to be defined.
CamId	Integer	Defines the cam disc ID (`CamID`) to which the segment should be assigned.
SegmentId	Integer	A unique number used to identify the segment in the cam disc selected with the `CamId` .

Parameters	data type	Meaning
Segment Type	MCTechnoCamSegment-Types (optional)	Type of curve shape for this segment relative to the centerline position.
	MCTechnoCamSegment-Types.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegment-Types.Poly3rd	curve shape: A third-degree polynomial.
	MCTechnoCamSegment-Types.Poly5th	curve shape: A fifth-degree polynomial.
	MCTechnoCamSegment-Types.Poly7th	curve shape: A seventh-degree polynomial.
	MCTechnoCamSegment-Types.Sine	curve shape: Sine function.
Segment Start	Double (optional)	The lead position from which this segment applies.
Option	MCTechnoCamOptions (optional)	Options for executing the segment.
	MCTechnoCamOp-tions.NoOption	The definition of the segment is accepted as stated.
	MCTechnoCamOp-tions.AutoAppend	This option is available starting with the second segment. It is intended to simplify the definition so that the offset of the sub-sequent axis for the segments does not need to be specified.
	MCTechnoCamOptions. User-Defined Master Cam Range	This option is available only in the first segment. Here, it is specified for the entire cam disc that the lead screw range of the cam disc is not equal to the module range of the lead screw. If the cam disc is to be stretched or compressed in the refer-ence axis direction during operation, the `UserDefinedMas-terRange` option must be selected here as well, even if the specified `MasterCamRange` corresponds to the modulo travel range of the reference axis.
Master Offset	Double (optional)	Offset of the master axis value, which is added to the current master axis position.
FollowerOffset	Double (optional)	Offset of the subsequent axis value, which is added to the function result.
Coefficient0	Double (optional)	Coefficients of the selected function according to the `segment type`. Not all coefficients are used for all `segment types` . **NOTICE! Please refer to the detailed information on poly-nomial formulas in Application Note 052.**
Coefficient1	Double (optional)	
Coefficient2	Double (optional)	
Coefficient3	Double (optional)	
Coefficient4	Double (optional)	
Coefficient5	Double (optional)	
Coefficient6	Double (optional)	
Coefficient7	Double (optional)	
MasterCam-Range	Double (optional)	If the `UserDefinedMasterCamRange` cam disc option is used, the non-standard master cam range must be defined here.

Syntax

```
tecGroup.Coupling.Cam.DefineSegment(xFollower,1,1, MCTechnoCamSegment-
Types.StraightLine, X1Offset, MCTechnoCamOptions.NoOption,-1.0 * X1Offset,
Y1Offset,0.0,1.0)
```

state administration

As long as the technology group is active, the function can be accessed in any state.

74.2.3 DefineSegmentAuto()

NOTICE

Risk of uncontrolled axis movement

Changes to the segment definitions take effect immediately.

- ► Avoid changing the segment definition while a cam disc is currently active.
- ► If it cannot be avoided, proceed with the utmost caution.

This function allows you to define, relative to the reference axis, the motion profile that should apply to a specific section of the cam disc. You can only define a segment if the cam disc has been created beforehand.

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated.

Difference between DefineSegment() and DefineSegmentAuto()

`DefineSegment()`: The polynomial segments are defined using the polynomial coefficients.

`DefineSegmentAuto()`: The polynomial segments are defined using initial and boundary conditions.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A follow-up object that must be a member of the technology group and for which a cam segment is to be defined.
CamId	Integer	Defines the cam disc ID (`CamID`) to which the segment should be assigned.
SegmentId	Integer	A unique number used to identify the segment in the cam disc selected by the `CamId`.
Segment Type	MCTechnoCamSegmentAutoTypes (optional)	Type of curve shape for this segment relative to the centerline position.
	MCTechnoCamSegmentAutoTypes.AutoPoly1st	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegmentAutoTypes.AutoPoly3rd	curve shape: A third-degree polynomial.
	MCTechnoCamSegmentAutoTypes.AutoPoly5th	curve shape: A fifth-degree polynomial.
	MCTechnoCamSegmentAutoTypes.AutoPoly7th	curve shape: A seventh-degree polynomial.
Segment Start	Double (optional)	The lead position from which this segment is valid.

Parameters	data type	Meaning
Option	MCTechnoCamOptions (optional)	Options for executing the segment.
	MCTechnoCamOptions.NoOption	The definition of the segment is accepted as stated.
	MCTechnoCamOptions.AutoAppend	This option is available starting with the second segment. It is intended to simplify the definition so that the offset of the subsequent axis for the segments does not need to be specified.
	MCTechnoCamOptions.User-Defined Master Cam Range	This option is available only in the first segment. Here, it is specified for the entire cam disc that the master axis region of the cam disc is not equal to the module region of the master axis. If the cam disc is to be stretched or compressed in the reference axis direction during operation, the `UserDefinedMasterRange` option must be selected here as well, even if the specified `MasterCamRange` corresponds to the modulo travel range of the reference axis.
FollowerOffset	Double (optional)	Offset of the subsequent axis value, which is added to the function result.
MasterRange	Double (optional)	Leitachsweg, which the segment crosses.
Follower Range	Double (optional)	The path that the segment traces.
vStart	Double (optional)	Initial and final conditions for the first, second, and third derivatives of the secondary axis position with respect to the primary axis position. Not all parameters are used for all `segment types`. **NOTICE! Please refer to the detailed information on derivations in Application Note 052.**
vEnd	Double (optional)	
aStart	Double (optional)	
aEnd	Double (optional)	
jStart	Double (optional)	
jEnd	Double (optional)	
MasterCam-Range	Double (optional)	If the `UserDefinedMasterCamRange` cam disc option is selected, the non-standard master cam range must be defined here.

Syntax

```
tecGroup.Coupling.Cam.DefineSegmentAuto(xFollower,1,1, MCTechnoCamSegmentAutoTypes.AutoPoli1st, X1Offset, MCTechnoCamOptions.NoOption, Y1Offset,0.0,1.0)
```

state administration

As long as the technology group is active, the function can be accessed in any state.

74.2.4 Activate()

Use this function to activate a cam disc in an active technology group. Activation is only possible after the cam disc has been created. Activation is, in turn, a prerequisite for coupling the cam disc axis.

To switch from one cam disc to another, activate the new (previously defined) cam disc using a switch condition. The secondary shaft remains coupled.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	A follow-up object that must be a member of the technology group and for which a cam disc should be activated.
CamId	Integer	Defines the cam disc ID (`CamID`) of the cam disc to be activated.
MasterOffset	Double (optional)	Defines the displacement of the cam disc in the lead angle direction.
FollowerOffset	Double (optional)	Defines the displacement of the cam disc in the follow-up axis direction.
MasterFaktor	Double (optional)	Defines the scale of the cam disc in the lead angle direction.
Follower Factor	Double (optional)	Defines the scaling of the cam disc in the axis direction.
Start Mode	MCTechnoCamStartModes (optional)	Defines the activation method.
	MCTechnoCamStart-Modes.Immediate	The cam disc engages immediately.
	MCTechnoCamStart-Modes.AtCamEnd	The cam disc is activated as soon as the end of the current cam disc is reached.
	MCTechnoCamStart-Modes.AtMasterPosition	The cam disc is activated when the reference axis position defined in the `Position` parameter is reached.
	MCTechnoCamStart-Modes.AtFollowerPosition	The cam disc is activated when the sequence axis position defined in the `Position` parameter is reached.
Position	Double (optional)	Defines the reference or follow-up axis position, depending on `the start mode`.

Syntax

```
tecGroup.Coupling.Cam.Activate(xFollower,1,0.0,0.0,1.0,1.0, MCTechnoCam-
StartModes.Immediate,0.0);
```

state administration

As long as the technology group is active, the function can be accessed in any state.

74.2.5 ReadSegment()

Use this function to read the data for a cam segment from a defined cam disc. The segment data is returned in a structure of type `CamSegmentData` .

Internally, the MCX kernel always works with polynomial coefficients, even if auto-polynomials were used to define the segment. For auto-polynomials, the `ReadSegment()` function therefore returns the values converted into polynomial coefficients.

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	The target object, which must be a member of the technology group, and for which the cam profile data is to be read.

Parameters	data type	Meaning
CamId	Integer	Defines the ID (`CamID`) of the cam disc whose segment data is to be read.
SegmentId	Integer	Defines the ID of the segment whose data is to be read.

The `CamSegmentData`structure, in which the data is stored, has the following parameters:

Parameters	data type	Meaning
Segment Type	MCTechnoCam Segment Types	Type of curve shape for this segment relative to the centerline position.
	MCTechnoCamSegment-Types.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegmentTypes.Poly3rd	curve shape: A third-degree polynomial.
	MCTechnoCamSegmentTypes.Poly5th	curve shape: A fifth-degree polynomial.
	MCTechnoCamSegmentTypes.Poly7th	curve shape: A seventh-degree polynomial.
	MCTechnoCamSegmentTypes.Sine	curve shape: Sine function
	MCTechnoCamSegmentTypes.Au-toPoly1th	curve shape: First-degree (linear) auto-polynomial
	MCTechnoCamSegmentTypes.Au-toPoly3rd	curve shape: Auto-polynomial of the third degree.
	MCTechnoCamSegmentTypes.Au-toPoly5th	curve shape: Auto-polynomial of the fifth degree.
	MCTechnoCamSegmentTypes.Au-toPoly7th	curve shape: Auto-polynomial of the seventh de-gree.
Segment Start	Double	The lead position from which this segment is valid.
Option	Int	Options for implementing the segment `0`: noOption `1`: AutoAppend `2`: User-Defined Master Cam Range
Master Offset	Double	Offset of the master axis value, which is added to the current master axis position.
FollowerOffset	Double	Offset of the subsequent axis value, which is added to the function result.
Coefficient	Array[0 .. 7] of Double	Coefficients of the function for this segment (al-ways expressed as polynomial coefficients).
MasterCam-Range	Double	For the first segment of a cam disc, the master shaft area of the cam disc is returned here. For each additional segment of a cam disc, the module range of the reference axis is returned here.

Syntax

```
SegmentData := tecGroup.Coupling.Cam.ReadSegment(xFollower,1,2);
```

state administration

As long as the active technology group is in operation, the function can be accessed in any state.

74.2.6 Evaluate()

Use this function to read the position and segment data from a cam disc. This function allows you to determine the secondary axis position and the corresponding segment associated with a primary axis position that you specify.

The data is returned in a structure of type `CamEvaluateData` .

When using the `AutoAppend` option in the `DefineSegment` or `DefineSegmentAuto` command, a correct value for the follower axis position cannot be read until after the cam disc has been started, since the follower axis offset is not calculated until after the cam disc has been activated.

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	The target object, which must be a member of the technology group, and for which the cam profile data is to be read.
CamId	Integer	ID (`CamID`) of the cam disc whose position and segment data are to be read.
Master's Degree	Double	The key position for which the corresponding data is to be determined.

The `CamEvaluateData` structure, in which the data is stored, has the following parameters:

Parameters	data type	Meaning
Follower Position	Double	Target position of the follower axis, calculated by the cam disc based on the position of the master axis that was passed to it.
Segment No.	Integer	The ID of the segment in which the position was determined.
MasterCamRange	Double	The leading axis range of the cam disc.

Syntax

```
CamPosData := tecGroup.Coupling.Cam.Evaluate(xFollower,1,1721.2);
```

state administration

As long as the active technology group is in operation, the function can be accessed in any state.

74.2.7 AnalyzeSegment()

This function allows you to read the data for a cam segment from a defined cam disc whose segments were created using the `AutoPoly` type. The function call returns an error if the cam disc was created using segments of a different type.

The segment data is returned in a structure of type `CamAnalyseData` .

This function can be used primarily to read the minimum and maximum values of the position (i.e., velocity, acceleration, and jerk) within a segment. **NOTICE! Please note that these figures are specific to the segment. To determine the actual minimum or maximum position, the follow-up axis offset must be added.**

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	The target object, which must be a member of the technology group, and for which the cam profile data is to be read.
CamId	Integer (optional)	ID (`CamID`) of the cam disc whose segment data is to be read.
SegmentId	Integer (optional)	The ID of the segment whose data should be read.
Option	MCTechnoCamAnalysisOptions (optional)	
	MCTechnoCamAnalysisOptions.AsCreated	Outputs the offsets in `CamAnalyseData.MasterOffset` and `CamAnalyseData.FollowerOffset` exactly as defined by the user.
	MCTechnoCamAnalysisOptions.AsActivated	Outputs the offsets in `CamAnalyseData.MasterOffset` and `CamAnalyseData.FollowerOffset` along with the additional activation offsets that the MCX calculates when the cam disc is activated.

The `CamAnalyseData`structure, in which the data is stored, has the following parameters:

Parameters	data type	Meaning
Segment Type	MCTechnoCam Segment Types	Type of curve shape for this segment relative to the centerline position.
	MCTechnoCamSegmentTypes.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegmentTypes.Poly3rd	curve shape: A third-degree polynomial.
	MCTechnoCamSegmentTypes.Poly5th	curve shape: A fifth-degree polynomial.
	MCTechnoCamSegmentTypes.Poly7th	curve shape: A seventh-degree polynomial.
	MCTechnoCamSegmentTypes.Sine	curve shape: Sine function
	MCTechnoCamSegmentTypes.AutoPoly1st	curve shape: First-degree (linear) auto-polynomial.
	MCTechnoCamSegmentTypes.AutoPoly3rd	curve shape: Auto-polynomial of the third degree.
	MCTechnoCamSegmentTypes.AutoPoly5th	curve shape: Auto-polynomial of the fifth degree.
	MCTechnoCamSegmentTypes.AutoPoly7th	curve shape: Auto-polynomial of the seventh degree.
	-1	An error has occurred, for example, because the segment was not defined as an `AutoPoly` . All other return values are no longer relevant.

Parameters	data type	Meaning
Segment Start	Double	The lead position from which this segment is valid.
Option	Integer	Options for executing the segment: `0`: noOption `1`: AutoAppend `2`: User-Defined Master Cam Range
MasterOffset	Double	Offset of the master axis value, which is added to the current master axis position.
FollowerOffset	Double	Offset of the subsequent axis value, which is added to the function result.
MinValue	Double	`[0]`: Minimum value of the 1st Derivation of position (velocity). `[1]`: Minimum value of the 2nd Derivation of position (acceleration). `[2]`: Minimum value of the 3rd Derivation of position (jerk).
MaxValue	Double	`[0]`: Maximum value of the 1st Derivation of position (velocity). `[1]`: Maximum value of the 2nd Derivation of position (acceleration). `[2]`: Maximum value of the 3rd Derivation of position (jerk).

Syntax

```
SegmentData := tecGroup.Coupling.Cam.AnalyzeSegment(xFollower,1,2);
```

state administration

As long as the active technology group is in operation, the function can be accessed in any state.

74.2.8 AnalyzeDerivatives()

This function allows you to read the data for a cam segment from a defined cam disc whose segments were created using the `AutoPoly` type. The function call returns an error if the cam disc was created using segments of a different type.

The segment data is returned in a structure of type `CamAnalyseDerivatesData` .

This function can be used primarily to read the minimum and maximum values of the position derivatives (i.e., velocity, acceleration, and jerk relative to the reference velocity) within a segment.

If the function is applied to the master axis or to a slave axis that is configured as an electric gear, an error is generated

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	The target object, which must be a member of the technology group, and for which the cam profile data is to be read.
CamId	Integer (optional)	The ID of the cam disc whose segment data is to be read.
SegmentID	Inteter (optional)	The ID of the segment whose data should be read.

The `CamAnalyseDerivatesData` structure, in which the data is stored, has the following parameters:

Parameters	data type	Meaning
Segment Type	MCTechnoCam Segment Types	Type of curve shape for this segment relative to the centerline position.
	MCTechnoCamSegment-Types.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegment-Types.Poly3rd	curve shape: A third-degree polynomial.
	MCTechnoCamSegment-Types.Poly5th	curve shape: A fifth-degree polynomial.
	MCTechnoCamSegment-Types.Poly7th	curve shape: A seventh-degree polynomial.
	MCTechnoCamSegment-Types.Sine	curve shape: Sine function
	MCTechnoCamSegment-Types.AutoPoly1th	curve shape: First-degree (linear) auto-polynomial.
	MCTechnoCamSegment-Types.AutoPoly3rd	curve shape: Auto-polynomial of the third degree.
	MCTechnoCamSegment-Types.AutoPoly5th	curve shape: Auto-polynomial of the fifth degree.
	MCTechnoCamSegment-Types.AutoPoly7th	curve shape: Auto-polynomial of the seventh degree.
	-1	An error has occurred, for example, because the segment was not defined as an `AutoPoly` . All other return values are no longer relevant.
Segment Start	Double	The lead position from which this segment is valid.
Option	Integer	Options for executing the segment: `0`: noOption `1`: AutoAppend `2`: User-Defined Master Cam Range
MasterOffset	Double	Offset of the master axis value, which is added to the cur-rent master axis position.
FollowerOffset	Double	Offset of the subsequent axis value, which is added to the function result.
MinValue	Array[0 .. 2] of Double	`[0]`: Minimum value of the 1st Derivation of position (velocity). `[1]`: Minimum value of the 2nd Derivation of position (acceleration). `[2]`: Minimum value of the 3rd Derivation of position (jerk).

Parameters	data type	Meaning
MaxValue	Array[0 .. 2] of Double	`[0]`: Maximum value of the 1st Derivation of position (velocity). `[1]`: Maximum value of the 2nd Derivation of position (acceleration). `[2]`: Maximum value of the 3rd Derivation of position (jerk).
MasterCam-Range	Double	The leading axis range of the cam disc.

Syntax

```
SegmentData := tecGroup.Coupling.Cam.AnalyzeSegmentDerivatives(xFol-
lower,1,2);
```

state administration

As long as the active technology group is in operation, the function can be accessed in any state.

75 MCTechno.Diagnostics

The **MCTechno.Diagnostics** function allows you to manage error handling for a technology group.

The Motion Control eXtended system distinguishes between *errors* and *messages*. Errors trigger a system response, whereas messages do not. Errors and messages are logged in separate buffers, from which they can be read. The buffer collects the events that have occurred since the last call to the `ClearErrors`() function (see ClearErrors() [▶ 40]).

The meanings of the error and message numbers are listed in the JetSym online help under “MotionControl Errors.”

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

75.1 Syntax for MCTechno.Diagnostics

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Verbundsbezeichnung>.Diagnostics.<Function or Access to Value>(optional
<Parameter>);
```

75.2 Features of MCTechno.Diagnostics

This function offers the following functions:

– ClearErrors()
– Software.ClearErrors()
– Software.GetErrorCode()
– Software.GetMessageCode()

75.2.1 ClearErrors(), Software.ClearErrors() – technology group

If errors occur, the functions acknowledge an error condition in the technology group. The entries in the error buffer are deleted.

Messages do not trigger an error response and therefore do not need to be acknowledged. For messages, these two functions cause the entries in the message buffer to be deleted.

transfer parameter

These functions do not have any transfer parameters.

Syntax

```
tecGroup.Diagnostics.ClearErrors();
tecGroup.Diagnostics.Software.ClearErrors();
```

ⓘ Info

Scope of the order

These commands clear all errors assigned by the group. These can be errors triggered by a group object, as well as errors reported by the group axes.

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state	
Blocked	–	Blocked	
Approved	–	Approved	
Working	–	Working	
Interrupted	–	Interrupted	
Error: Approved	√	Approved	The system will only switch to this state once the cause of the error has been resolved. Otherwise, the error condition may persist or recur immediately.
Error: blocked	√	Blocked	

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

75.2.2 Software.GetErrorCode()

This function returns the error number at the error buffer location specified by `BufferIndex` . If no `BufferIndex` is passed, the number of the most recently recorded error is returned (`BufferIndex` = `0`).

transfer parameter

Parameters	data type	Meaning
BufferIndex	Integer (optional)	The `BufferIndex` parameter specifies the position in the error buffer from which the error number is to be read. The maximum value of the `BufferIndex` parameter can be retrieved via the `ErrorCount` property.

Syntax

```
tecGroup.Diagnostics.Software.GetErrorCode();
```

state administration

The function can be read in any state.

75.2.3 Software.GetMessageCode()

This function returns the message number at the message buffer location specified by `BufferIndex` . If no `BufferIndex` is provided, the number of the most recently recorded message is returned (`BufferIndex` = `0`).

transfer parameter

Parameters	data type	Meaning
BufferIndex	Integer (optional)	The `BufferIndex` specifies the position in the message buffer from which the message number is to be read. The maximum value of the `BufferIndex` parameter can be retrieved via the `MessageCount` property.

Syntax

```
tecGroup.Diagnostics.Software.GetMessageCode();
```

state administration

The function can be read in any state.

75.3 Properties of MCTechno.Diagnostics

This function offers the following properties:

– Software.ErrorCount
– IsErrorPending
– Software.IsErrorPending
– Software.MessageCount
– IsMessagePending
– Software.IsMessagePending

75.3.1 Software.ErrorCount

Description

property	data type	Meaning
Software.ErrorCount	Int	Reads the number of errors logged since the last time the `ClearErrors()` or `Software.Clear.Errors()` function was called, or since the controller was restarted.

Syntax

```
ErrorCount := tecGroup.Diagnostics.Software.ErrorCount;
```

state administration

The property can be read in any state.

75.3.2 IsErrorPending, Software.IsErrorPending

Description

data type	Meaning
Bool	Check whether there is an error in the MCX system. This is equivalent to comparing `ErrorCount` to 0.

Syntax

```
If tecGroup.Diagnostics.IsErrorPending then
...
End_if;
```

state administration

The property can be queried in any state.

75.3.3 Software.MessageCount

Description

data type	Meaning
Integer	Reads the number of logged messages since the last time the `ClearErrors()` or `Software.ClearErrors()` function was executed, or since the controller was restarted.

Syntax

```
MessageCount := tecGroup.Diagnostics.Software.MessageCount;
```

state administration

The property can be read in any state.

75.3.4 IsMessagePending, Software.IsMessagePending

Description

data type	Meaning
Bool	Check whether there is a message in the MCX system. Dies ist gleichbedeutend mit einem Vergleich von `MessageCount` mit `0`.

Syntax

```
If tecGroup.Diagnostics.IsMessagePending then
...
End_if;
```

state administration

The property can be queried in any state.

75.4 Example task for MCTechno.Diagnostics

Here is an example task for a single axis that includes various Motion API functions:

```
Var
    TecError: Array[10] of INT; //Variables for displaying the error buffer
    i: INT; //Loop variable
    AcknolegeError: Bool; // Auxiliary flag for an acknowledgment condition

End_Var;

Task Sample_Error Autorun

Loop

    //Clear error variables for display
    For i := 0 To 9 Do
        TecError[i] := 0;
    End_For;
    AcknowledgeError := False;

    If tecGroup.Diagnostics.IsErrorPending is true, continue;
```

```
        Taskbreak Automatic;
        // Add additional error handling here

        //Store errors in variables, e.g., for display purposes
        For i := 0 To tecGroup.Diagnostics.Software.ErrorCount- 1 Do
            TecError[i] := tecGroup.Diagnostics.Software.GetErrorCode(i);
        End_For;

    // Bugs have been fixed
    When AcknowledgeError , continue;

        tecGroup.Diagnostics.ClearErrors();
        Automatictask restart ;
End_Loop;

End_Task;
```

76 MCTechno.Info

The **MCTechno.Info** function allows you to identify the group objects in MotionControl.

This information was entered by the user in the Hardware Manager and motion setup within the project.

76.1 Syntax for MCTechno.Info

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Technologieverbundbezeichnung>.Info.<Eigenschaft>;
```

76.2 Properties of the function

This function offers the following properties:

– Name
– ObjectID
– ClassID

76.2.1 Name

Description

data type	Meaning
String	Retrieves the name of the technology group defined in the hardware configuration. This is helpful if you want to use the name in the trace or on an external display.

Syntax

```
TechnoName := Techno.Info.Name;
```

`TechnoName` contains the string `"Techno"`.

state administration

The property can be read in any state.

76.2.2 ObjectID

Description

data type	Meaning
Integer	Retrieves the internal control number of the technology group. This property is typically only required in applications that use the MC Manager's features. One possible use case is described in the corresponding section MCManager (MCMgr) [▶ 401] .

Syntax

```
Composite ID := TechnoGroup.Info.ObjectID;
```

state administration

The property can be read in any state.

76.2.3 ClassID

Description

data type	Meaning
MCClassIDs	Queries the class of an MC object. An MC object can be an axis, a technology group, or a path group. This property is typically only required in applications that use the MC Manager's features. One possible use case is described in the corresponding section MCManager (MCMgr) [▶ 401] .

Syntax

```
ClassID := tecGroup.Info.ClassID;
```

state administration

The property can be read in any state.

76.3 Example Task MCTechno.Info

Here is an example task for a single axis that includes various Motion API functions:

After the example task has been executed, the trace displays whether each technology group is in the **"running"** state or not. The groups are listed by their group names.

```
function TraceRunningState(mcTechnoGroup: pointer to MCTechno)
    if mcTechnoGroup.State.IsRunning then
        trace(mcTechnoGroup.Info.Name + 'is running$n');
    else
        trace(mcTechnoGroup.Info.Name + 'is not running$n');
    end_if;
end_function;

task Sample1 // example task
var
    nTechnoIndex:    int;
end_var;
    for nTechnoIndex := 0 to mcMgr.TechnoCount - 1 , incrementing by 1 , do
        TraceRunningState(mcMgr.TechnoList[nTechnoIndex]);
    end_for;
end_task;
```

77 MCTechno.Mechanism.Slope

The **MCTechno.Mechanism.Slope** function allows you to read or adjust the calculated properties of the current or next movement.

A positioning movement consists of the following phases:

1. acceleration slope
2. Driving at a constant speed
   This phase occurs only if the speed can be reached using the specified MovePtp parameters. Otherwise, the acceleration slope transitions directly into the deceleration slope (triangular mode).
3. deceleration slope

In addition, no movement can occur if the axis is at its destination and no further positioning has been initiated.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

77.1 Syntax for MCTechno.Mechanism.Slope

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Mechanism.Slope.<Access to Value>
```

77.2 Functions of MCTechno.Mechanism.Slope

This function offers the following functions:

– GetSlopeType()
– SetSlopeType()
– GetSpeed()
– SetSpeed()
– GetAcceleration()
– SetAcceleration()
– GetDeceleration()
– SetDeceleration()

77.2.1 GetSlopeType(), SetSlopeType()

Use these functions to view or modify the slope profile during the acceleration phase and the deceleration phase of the composite axis specified in the first parameter. **NOTICE! This setting applies to all positioning movements of the MCX axis until the value is changed.**

The `GetSlopeType()` function returns a value of type `MCTechnoSlopeTypes` . All you need to do is pass it a pointer to an axis object that is part of the technology group.

transfer parameter

Parameters	data type	Meaning
Axis	pointer to MCAxis	Axis object (which must be a member of the technology group) whose slope profile is to be read or modified
Slope Type	MCTech-noSlopeTypes	Shape of the acceleration and deceleration slope.
	AxisSlopeTypes.Sine (Default)	This setting calculates a motion that produces relatively little jerk, meaning the motion is smoother. Since the acceleration value passed to the motion command is only reached at the midpoint of the slope, the slope duration and length are longer than with the linear setting. The motion follows a sinusoidal path during the acceleration phase and a sin²-shaped path during the deceleration phase. The movement is limited to a certain extent.
	AxisSlopeTypes.Lin-ear	This setting allows for motion with the lowest possible maximum ac-celeration while maintaining the same slope time (and duration) as the `"sine"` setting. However, the jolt during the transition from rest to the acceleration phase and from the acceleration phase to the phase of maximum speed is infinite. The motion is constant during the acceler-ation phase and sinusoidal during the deceleration phase. The movement occurs without any jerk limitation.

Syntax

```
RampType := tecGroup.Mechanism.Slope.GetSlopeType(axLeitachse);
tecGroup.Mechanism.Slope.SetSlopeType(axLeitachse,
MCTechnoSlopeTypes.Linear);
```

state administration

The function can be called from any state.

If the slope profile is changed using `SetSlopeType()` during a positioning operation (in the **"Running"**state), the change will not take effect until the next positioning operation.

77.2.2 GetSpeed(), SetSpeed()

Description

Reads or writes the value for the maximum speed used to generate the position setpoint for a positioning profile. This value applies to the composite axis specified in the first parameter.

This setting applies to all positioning movements of the composite axis until the value is changed.

This value is typically passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` , or in `MovePtp.NewSpeed()` and `MoveVelocity.NewSpeed()` , where a change is applied to a movement that is already in progress.

It's different here: Regardless of the state in which the change to the `speed` value is passed, the new value does not take effect until the next movement.

Parameters	data type	Meaning
Axis	pointer to MCAxis	An axis object that must be a member of the technology group and whose profile speed is to be read or modified.
Speed	Double	Maximum deceleration in the positioning profile.

Syntax

```
ProfileSpeed := tecGroup.Mechanism.Slope.GetSpeed(axLeitachse);
tecGroup.Mechanism.Slope.SetSpeed(axLeitachse, 3000.0);
```

state administration

The function can be called from any state.

If the profile speed is changed using `SetSpeed()` during a positioning operation (in the **"Running"**state), the change will not take effect until the next positioning operation.

77.2.3 GetAcceleration(), SetAcceleration()

Description

Reads or writes the value for the maximum acceleration used to generate the position setpoint for a positioning profile. This value applies to the composite axis specified in the first parameter.

This setting applies to all positioning movements of the composite axis until the value is changed.

This value is typically passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` , or in `MovePtp.NewAcceleration()` and `MoveVelocity.NewAcceleration()`

Parameters	data type	Meaning
Axis	pointer to MCAxis	An axis object that must be a member of the technology group and whose profile acceleration is to be read or modified.
Acceleration	Double	Maximum acceleration in the positioning profile.

Syntax

```
ProfileAcceleration := tecGroup.Mechanism.Slope.GetAccelera-
tion(axLeitachse);
tecGroup.Mechanism.Slope.SetAcceleration(axLeitachse, 5500.0);
```

state administration

The function can be called from any state.

If the profile acceleration is changed using `SetAcceleration()` during a positioning operation (in the **"Running"**state), the change will not take effect until the next positioning operation.

77.2.4 GetDeceleration(), SetDeceleration()

Description

Reads or writes the value for the maximum delay used to generate the position setpoint for a positioning profile. This value applies to the composite axis specified in the first parameter.

This setting applies to all positioning movements of the composite axis until the value is changed.

This value is typically passed as a parameter in the `MovePtp.Start()` and `Move.Velocity.Start()` functions, or in the `MovePtp.NewDeceleration()` and `MoveVelocity.NewDeceleration()` functions.

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and whose profile delay is to be read or modified.
Deceleration	Double	Maximum deceleration in the positioning profile.

Syntax

```
ProfileDeceleration := tecGroup.Mechanism.Slope.GetDecelera-
tion(axLeitachse);
tecGroup.Mechanism.Slope.SetDeceleration(axLeitachse,12000.0);
```

state administration

The function can be called from any state.

If the profile deceleration is changed using `SetDeceleration()` during positioning (**Running**state), the change will not take effect until the next positioning operation.

77.3 Properties of MCTechno.Mechanism.Slope

This function offers the following properties:

– SlopeState
– Is Accelerating
– IsAtConstantSpeed
– IsDecelerating
– IsStopped

77.3.1 SlopeState

Description

data type	Meaning
MCTechnoSlopeStates	Reads the current phase of motion (acceleration, constant speed, deceleration, or standstill) of the path setpoint for the path group. The slope status of the master axis is returned. If the technology group is inactive, **"paused"** is returned. With this property, you can, for example, delay the activation of an action in the machine until the group has exited the acceleration phase.
MCTechnoSlopeStates.Accelerating	The group is on the acceleration slope.
MCTechnoSlopeStates.AtConstantSpeed	The group is traveling at a constant speed.
MCTechnoSlopeStates.Decelerating	The group is on the deceleration slope.
MCTechnoSlopeStates.Stopped	The group is down.

Syntax

```
SlopeState := tecGroup.Mechanism.Slope.SlopeState;
```

state administration

The property can be read in any state.

77.3.2 IsAccelerating

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the master axis of the technology group is within the acceleration slope.

Syntax

```
If tecGroup.Mechanism.Slope.IsAccelerating then …
```

state administration

The property can be queried in any state.

77.3.3 IsAtConstantSpeed

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the master axis of the technology group is in the constant-speed phase.

Syntax

```
If tecGroup.Mechanism.Slope.IsAtConstantSpeed then …
```

state administration

The property can be queried in any state.

77.3.4 IsDecelerating

Description

data type	Meaning
Bool	Check whether the calculated setpoint for the master axis of the technology group is within the deceleration slope.

Syntax

```
If tecGroup.Mechanism.Slope.IsDecelerating then …
```

state administration

The property can be queried in any state.

77.3.5 IsStopped

Description

data type	Meaning
Bool	Check whether the calculated target value for the main axis of the technology group is in the idle phase.

Syntax

```
If tecGroup.Mechanism.Slope.IsStopped then …
```

state administration

The property can be queried in any state.

78 MCTechno Members

The **MCTechno.Members** function allows you to view the composition of a technology group.

For very specific cases, you can also add or remove composite axes from the STX program. Since these cases are recommended only in exceptional circumstances, these commands are not described here. Please contact [Bucher Automation AG] if you would like to use these features.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

78.1 Syntax for MCTechno.Members

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Members.<Access to Value>(optional <Parameter 1>)
```

78.2 Properties of MCTechno.Members

This function offers the following properties:

- Follower Count
- Followers[]
- Follower Count
- FollowerCam[]
- FollowerGearCount
- FollowerGear[]
- Master's degree

78.2.1 Follower Count

Description

data type	Meaning
Integer	Reads the number of dependent objects in the technology group.

Syntax

```
NumberOfFollowers := tecGroup.Techno.Members.FollowerCount;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.2 Follower[s]

Description

data type	Meaning
Pointer to MCAxis	Reads the axis object with number *n* in the list of secondary axes of the technology group. The list starts at `0` and goes up to `FollwerCount-1`.

Syntax

```
FollowerAxisName := tecGroup.Members.Follower[1].Info.Name;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.3 Follower Count

Description

data type	Meaning
Integer	Reads the number of child objects in the technology group that are defined as cam discs.

Syntax

```
NumberOfCurveAxes := tecGroup.Techno.Members.FollowerCamCount;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.4 FollowerCam[n]

Description

data type	Meaning
Pointer to MCAxis	Reads the axis object with number *n* in the list of cam disc sequence axes of the technology group. The list starts at `0` and goes up to `FollwerCamCount-1`.

Syntax

```
CurveAxisObject := tecGroup.Members.FollowerCam[1].Info.ObjectID;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.5 FollowerGearCount

Description

data type	Meaning
Integer	Reads the number of child objects in the technology group that are defined as electric gears.

Syntax

```
numberOfGears := tecGroup.Techno.Members.FollowerGearCount;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.6 FollowerGear[n]

Description

data type	Meaning
Pointer to MCAxis	Reads the axis object with number *n* in the list of slave axes defined as an electric gear within the technology group. The list starts at `0` and goes up to `FollwerGearCount-1`.

Syntax

```
elGearObject := tecGroup.Members.FollowerGear[0].Info.ObjectID;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.7 Master's

Description

data type	Meaning
Pointer to MCAxis	Reads the axis object that serves as the main axis of the technology group.

Syntax

```
LeitAchsName := tecGroup.Members.Master.Info.Name;
```

state administration

The property can be queried in any state of the technology group, including the **inactive** state.

79 MCTechno.MoveHalt

The **MCTechno.MoveHalt** function allows you to stop a moving axis within a technology group using a slope.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

79.1 Syntax for MCTechno.MoveHalt

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveHalt.Start.(optional <Parameter 1, Parameter 2>);
```

79.2 Features of MCTechno.MoveHalt

This function offers the following functions:

– Start()

79.2.1 Start()

This function immediately executes the stop command on the axis specified in the first parameter. The parameters passed define how the axis should stop.

Please follow these rules:

– If the function is executed on the master axis, all axes will stop according to their coupling type. All axes remain coupled.

– If the function is executed on a decoupled secondary axis, that axis will stop according to the parameters provided.

– For a coupled secondary axis, the function acts on the possible superimposed motion. The movement of the slave axis is not affected by the coupling with the master axis.
To decouple and stop a coupled follow-up axis, use the function `MCTechno.Coupling.Decouple(axFollowUpAxis, MCTechnoDecoupleModes.Immediate_Stop, , DecelerationRamp)`.

– The `Deceleration` parameter is entered in the user units that are predefined for the axis.
Adjusting the maximum delay using this parameter changes the value in the axis object. For example, if no value is passed for the `Deceleration` parameter in the next `MovePtp.Start`command, the value specified here will be used.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

transfer parameter

Parameters	data type	Meaning
Axis	pointer to MCAxis	An axis object that must be a member of the technology group and should be encouraged to join.
MCAxisHoldMode	Enum MCAxisHalt-Modes (optional)	Behavior of the axis after it has been slowed down using a de-celeration slope:
	MCAxisHaltModes.Nor-mal	Once the slope has been completed, the axis comes to a stop under position control.
	MCAxisHaltModes.At-ActualPosition	After the stop slope is complete, the axis returns to the posi-tion it was in when the `MoveHalt.Start()` function was called. The reverse movement is performed using the param-eters set for this axis object.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the deceleration slope. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.

Syntax

```
tecGroup.MoveHalt.Start(axLeitachse, MCTechnoHaltModes.Normal,50000);
tecGroup.MoveHalt.Start(axLeitachse,
MCTechnoHaltModes.AtActualPosition,50000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	–		Blocked
Approved	–		Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

When checking whether the stop is complete, only the **slope status** should be taken into account.

The **operational status** of the group is the sum of the operational statuses of the group axes. As soon as one member moves, the operational state changes to **"Running"** and the **"Enabled"** state is only reached when all members are stationary. The state can therefore be ambiguous depending on the application.

slope status of the lead axis:

The group itself also has slope status. This is the slope status of the lead axis! Therefore, this must also be taken into account when querying the slope status while positioning the lead axis. The slope status of the master axis, as a single axis, remains set to **"Stopped"** while the group is active and therefore cannot be used.

```
// master axis
tecGroup.MoveHalt.Start(axLeitachse, MCTechnoHaltModes.Normal, 1000.0);
when tecGroup.Mechanism.Slope.IsStopped , continue;
// Secondary axis
tecGroup.MoveHalt.Start(axFolgeachse, MCTechnoHaltModes.Normal, 1000.0);
when axFolgeachse.Mechanism.Slope.IsStopped , continue;
```

80 MCTechno.MovePtp

The **MCTechno.MovePtp** function allows you to initiate and configure the point-to-point positioning of an axis within a technology group.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

80.1 Syntax for MCTechno.MovePtp

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MovePtp.<Function or Access to Value>(optional <Parameter 1, Parameter 2, Parameter n>);
```

80.2 Features of MCTechno.MovePtp

This function offers the following functions:

- Start()
- StartNewTargetPosition()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()
- GetPositioningMode()
- SetPositioningMode()
- GetTargetPosition()
- SetTargetPosition()
- GetTargetWindow()
- SetTargetWindow()
- StartSuperPose()
- StartNewTargetPositionSuperPose()

80.2.1 Start()

This function initiates point-to-point positioning of an axis in a technology group. The axis begins its movement immediately or applies the data to the current position.

Since all parameters required for positioning can be specified in a single command, this is the most concise way to fully configure and initiate a positioning movement.

If the function is executed on the master axis, all coupled slave axes move accordingly based on their linkage function.

If the function is executed on a coupled secondary axis, that axis is first uncoupled and then moved according to the function's parameters. A sub-axis that was originally coupled is not automatically recoupled once the movement is complete.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Positioning in unbuffered mode (default)

Depending on whether the axis is moving or not, the following behavior occurs:

– Calling the function in the **enabled**state: The movement begins.

– Calling the function in the **running**state: The movement parameters are applied to the current OR subsequent movement, as shown in the table below.

Parameters	data type	Meaning	Behavior when calling a function while in the "running" state
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.	
Positioning Mode	MCTechno Positioning Modes (optional)	Defines the positioning mode (see Get/SetPositioningMode [▶ 464]).	The mode takes effect immediately.
TargetPosition	Double (optional)	Defines the target position at which the axis should be located after positioning has been completed.	– If the new target position can be reached while adhering to the maximum values for speed, acceleration, deceleration, and jerk, the system moves directly to that position using the new motion parameters. – Otherwise, the current positioning operation will be completed, and the newly programmed one will be initiated immediately.
Speed	Double (optional)	Defines the speed at which the axis should be positioned when it is not in the slope (ac-celeration, deceleration) phase. The value must be greater than `0` .	If a new positioning operation is initiated while the previous one is still in progress (running state), the new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration slope parameters.

Parameters	data type	Meaning	Behavior when calling a function while in the "running" state
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start slope during positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.	The speed ramps are implemented according to the new acceleration and deceleration values. – If the motion is currently in such a slope, intermediate forms occur that do not correspond to the selected slope type. – Transitions are always performed under the control of the jerk limiter.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop slope while positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.	
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change to the target window is immediately applied to the current positioning.

Positioning phases

A positioning movement consists of the following phases:

1. acceleration slope
2. Constant speed
   This phase occurs only if the speed can be reached using the specified MovePtp parameters. Otherwise, the acceleration slope transitions directly into the deceleration slope (triangular mode).
3. deceleration slope

Types of ramps

The slope profiles for the acceleration and deceleration phases are specified by the axis parameter MCTechno.Mechanism.Slope.SetSlopeType (type MCTechnoSlopeTypes). There are two possible settings here:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration;
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = quadratic acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
tecGroup.MovePtp.Start(axLeitachse, AxisPositioningModes.AbsNor-
mal,20000,500,1000,1000,0.5);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

– The **state of the group** indicates whether the setpoint generator for all group axes is generating a movement or has been paused.

```
When tecGroup.State.IsEnabled Continue; //The coupled axes are in position
control mode and are stationary.
When tecGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

– If you want to check whether a specific position has been reached just before the destination, use the **destination window** . The query is only possible for the specific axis object of the composite axis.

```
When axLeitachse.IsInTargetWindow Continue;
```

See also

- MCAxis.Buffer [▶ 32]
- MCAxis.Mechanism.Slope [▶ 134]

80.2.2 StartNewTargetPosition()

This function initiates point-to-point positioning of a composite axis within the technology group. The axis begins moving immediately or applies the data to adjust the current position.

This function is a simplified version of the `Start()` function, which passes only the relevant position parame-ters. The speed values and slope settings for the axis are applied.

This section describes the specifics of the `StartNewTargetPosition()` function. For further information, please refer to the description of the Start() [▶ 457] function in the technology group.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.
– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning	Behavior when calling a function in the "Running" state
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.	
Positioning Mode	MCTechnoPositioning-Modes (optional)	The PositioningMode (see Get/SetPositioningMode [▶ 464]).	The mode takes effect immediately.
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change to the target window is immediately applied to the current positioning.

Syntax

```
tecGroup.MovePtp.StartNewTargetPosition(axLeitachse, AxisPositioning-
Modes.AbsNormal,20000,0.5);
```

80.2.3 NewSpeed()

The function sets a new value for the speed of point-to-point positioning for a composite axis in a technology group. The following applies:

– The value must be greater than `0` .
– If the axis is already in motion (**Running** state), the new speed value is applied immediately. When an axis is in motion, the transition from the current speed to the new speed occurs at the accelera-tion rate set for the acceleration slope. This is also the case when the speed decreases.
– The slope type cannot be changed during positioning.
– Jerk limiting is active for all transitions if the slope type is set to *sin* ².

ⓘ Info

Bring the axis to a stop

To bring an axis to a stop, use only the `MoveHalt` or `Stop` commands. It is **not** possible to stop an axis by setting its speed to `0` .

transfer parameter

Parameters	data type	Meaning	Behavior when calling a function while in the "Running" state
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.	
Speed	Double	Defines the speed at which the axis should be positioned when it is not in the slope (acceleration, deceleration) phases.	The new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration slope parameters.

Syntax

```
tecgroup.MovePtp.NewSpeed(axMainAxis,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.4 NewAcceleration()

The function sets a new value for the maximum acceleration of a point-to-point positioning operation for a composite axis within a technology group.

The following applies:

– An active acceleration slope is not modified.

– The new value will be applied during the next PtP positioning.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start slope dur-ing positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.

Syntax

```
tecGroup.MovePtp.NewAcceleration(axLeitachse,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working

Initial state	Allowed	Final state
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.5 NewDeceleration()

The function sets a new value for the maximum delay of point-to-point positioning on an MCX axis.

The following applies:

– The value is also used during an ongoing PtP positioning if the target can still be reached with the new parameter.

– Otherwise, the value will be applied during the next PtP positioning.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop slope while positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.

Syntax

```
tecGroup.MovePtp.NewDeceleration(axLeitachse,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.6 GetPositioningMode(), SetPositioningMode()

Description

These functions allow you to read the currently active positioning mode of a composite axis within a technology group or set a new value for that axis.

The following applies:

– A new value takes effect during the next point-to-point positioning, even if the change is made while positioning is in progress.
– After power-up, the final target position is calculated based on the actual position at the time of power-up.

Parameters	data type	Meaning
Axis	Link to MCAcis	An axis object that must be a member of the technology group and is to be moved.

Parameters	data type	Meaning
Positioning Mode	MCTechnoPositioningMode	Type of PtP movement.
	MCTechnoPositioningMode.AbsNormal	Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement for positioning is determined by the last position and the new target position.
	MCTechnoPositioningMode.RelativeTarget	Moves the axis by the distance specified in the `TargetPosition` parameter. The target position is calculated by adding the value in `TargetPosition`to the last target position. The direction of travel is determined by the sign of the `TargetPosition`parameter.
	MCTechnoPositioning-Mode.RelActual	Moves the axis by the distance specified in the `TargetPosition` parameter. The target position is calculated by adding the value in `TargetPosition`to the actual position at the start of the positioning process. The direction of travel is determined by the sign of the `TargetPosition`parameter.
	MCTechnoPositioningMode.AbsModuloPos	Applies only to modulo axes. Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement during positioning is always **positive**. In this case, the modulo wrap can be ignored if the new target position is less than the last target position.
	MCTechnoPositioningMode.AbsModuloNeg	Applies only to modulo axes. Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement during positioning is always **negative**. In this case, the modulo wrap can be ignored if the new target position is greater than the last target position.
	MCTechnoPositioningMode.AbsModuloAuto	Applies only to Modulo axes. – Positioning starts when the axis is paused: Moves the axis to the position specified in the `TargetPosition` parameter. It is approached **via the shortest path** relative to the reference position. The direction of movement for positioning is determined by the shortest path to the destination. – Positioning starts while the axis is moving: Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement for the positioning is determined by the direction of movement of the current positioning.
	MCTechnoPositioningMode.AbsModuloKeepDirection	Applies only to Modulo axes. Moves the axis to the position specified in the `TargetPosition` parameter. It is moved to this position relative to the reference position. The direction of movement for the positioning is determined by the direction of movement of the previous positioning.

Syntax

```
Leitach'sPositioningMode := tecGroup.MovePtp.GetPositioning-
Mode(axLeitachse);
tecGroup.MovePtp.SetPositioningMode(axFollowAxis,
MCTechnoPositioningModes.RelTarget);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.7 GetTargetPosition(), SetTargetPosition()

Description

These functions allow you to read the current value of `TargetPosition` for a composite axis in a technology group or write a new value for that axis. If the function is called while the system is in the **"Running"** state, the motion parameters take effect when the next point-to-point positioning operation begins.

Parameters	data type	Meaning
Axis	Link to MCAcis	An axis object that must be part of the technology group and is to be moved.
TargetPosition	Double	Property used to determine the target position based on the `PositioningMode` property.

Syntax

```
Target := tecGroup.MovePtp.GetTargetPosition(axFolgeachse);
tecGroup.MovePtp.SetTargetPosition(axLeitachse,12000);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working

Initial state	Allowed	Final state
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.8 GetTargetWindow(), SetTargetWindow()

Description

These functions allow you to read the current value of the target window setting for a composite axis in a technology group or to write a new value for that axis. This will take effect when the next point-to-point positioning begins.

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
TargetWindow	Double	Defines when the `IsInTargetWindow` status becomes active in the target position area of a positioning operation.

Syntax

```
TargetWindow := tecGroup.MovePtp.GetTargetWindow(axLeitachse);
tecGroup.MovePtp.SetTargetWindow(axFolgeachse,0.7);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.9 StartSuperPose()

ⓘ Info

APN 51 – Superimposed Positioning

Please refer to the detailed information in **Application Note 051** regarding the superimposed motion of a secondary axis in an electric gear.

This function initiates superimposed point-to-point positioning of a secondary axis in a technology group that is connected via an electric gear.

The following guidelines apply:

– If the function is executed on a coupled secondary axis that is configured as an electric gear, the superim-posed motion is initiated.

– If the function is executed on a decoupled slave axis that is configured as an electric gear, or on the mas-ter axis, a motion is initiated for that axis that behaves as if it had been executed using the Start() [▶ 457] function. For more information, please refer to the linked section.

– If the function is executed on a secondary axis that is configured as a cam disc, a message is displayed.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Positioning Mode	MCTechnoPositioning-Modes (optional)	Defines the positioning mode (see Get/SetPositioningMode [▶ 464]).
TargetPosition	Double (optional)	Defines the target position at which the axis should be positioned after the positioning operation has been completed.
Speed	Double (optional)	Defines the speed at which the axis should be positioned when it is not in the slope (acceleration, deceleration) phases. The value must be greater than `0` .
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start slope during positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration through-out.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop slope while positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration through-out.
TargetWindow	Double (optional)	Defines the width of the target window for positioning.

Syntax

```
tecGroup.MovePtp.StartSuperPose(axElGetriebe.AxisPositioningModes.AbsNor-
mal,200,500,1000,1000,0.5);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

80.2.10 StartNewTargetPositionSuperPose()

ⓘ Info

APN 51 – Superimposed Positioning

Please refer to the detailed information in **Application Note 051** regarding the superimposed motion of a secondary axis in an electric gear.

This function is a shortened version of the `StartSuperPose()` function. Both functions initiate superimposed point-to-point positioning of a secondary axis within a technology group, which is coupled via an electric gear. With this variant, only the parameters relevant to the item will be passed. The values for speed and slope settings are applied as they were last set for the axis.

The following guidelines apply:

– If the function is executed on a coupled secondary axis that is configured as an electric gear, the superim-posed motion is initiated.

– If the function is executed on a decoupled slave axis configured as an electric gear or on the master axis, a motion is initiated for that axis that behaves as if it had been executed using the StartNewTargetPosition() [▶ 460] function. For more information, please refer to the linked section.

– If the function is executed on a secondary axis that is configured as a cam disc, a message is displayed.

transfer parameter

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Positioning Mode	MCTechnoPositioning-Modes (optional)	Defines the positioning mode; see GetPositioningMode(), SetPositioningMode() [▶ 464].

Parameters	data type	Meaning
TargetWindow	Double (optional)	Defines the width of the target window for positioning.

Syntax

```
tecGroup.MovePtp.StartTargetPositionSuperPose(axElGear, AxisPositioning-
Modes.AbsNormal,200,0.5);
```

The details regarding state management and state monitoring are identical to those of the Start() [▶ 457]function.

81 MCTechno.MoveVelocity

The **MCTechno.MoveVelocity** function allows you to start and configure the continuous positioning of an MCX axis within a technology group.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 54, 55, and 57** for procedures and example projects related to axis control. These are available for download on the Bucher Automation website.

81.1 Syntax for MCTechno.MoveVelocity

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.MoveVelocity.<Function or Access to Value>(optional
<Parameter 1, Parameter 2, Parameter n>);
```

81.2 Functions of MCTechno.MoveVelocity

This function offers the following functions:

- Start()
- Reverse()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()
- GetDirection()
- SetDirection()
- StartSuperPose()
- ReverseSuperPose()

81.2.1 Start()

This function initiates continuous positioning of an axis within a technology group. The axis begins its movement immediately or applies the data to the current position.

The following applies:

- If the function is executed on the **master axis**, all coupled slave axes move accordingly based on their link-age function.
- If the function is executed on a coupled **secondary axis**, that axis is first unlinked and then moved accord-ing to the function's parameters. A sub-axis that was originally coupled is not automatically relinked once the movement is complete.
- The axis accelerates at the current or programmed slope rate and continues to move at the current or programmed `speed` until it is stopped by a hold or stop command. Continuous positioning is position-controlled, so that a position tracking error is also monitored during movement.

transfer parameter

Please follow these rules:

- All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Positioning in unbuffered mode

Depending on whether the axis is moving or not, the following behavior occurs:

– Calling the function in the **"Enabled"** state: The movement begins.

– Calling the function in the **running** state: The movement parameters are applied to the current OR subsequent movement according to the table below.

Parameters	data type	Meaning	Behavior when calling a function while in the "Running" state
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.	
Directions	Directions (optional) – Directions.Positive – Directions.Negative	Defines the direction of movement (see GetDirection(), SetDirection() [▶ 478]).	The new direction parameters are applied immediately. If the new direction is opposite to the current direction of motion, a reversal operation is performed using the acceleration ramps.
Speed	Double (optional)	Defines the speed at which the axis should be positioned when it is not in the slope (acceleration, deceleration) phases. The value must be greater than `0` .	If a new positioning operation is initiated while the previous one is still in progress (**Running** state), the new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration slope parameters.

Parameters	data type	Meaning	Behavior when calling a function while in the "Running" state
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start slope during positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration through-out.	The speed ramps are implemented according to the new acceleration and deceleration values. – If the motion is currently in such a slope, intermediate forms occur that do not correspond to the selected slope type. – Transitions are always performed under the control of the jerk limiter.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop slope for the next `MoveHalt.Stop()` command during positioning, unless another value is specified there. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration through-out.	

Types of ramps

The slope profiles for the acceleration and deceleration phases are specified by the axis parameter `MCTechno.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (see MCAxis.Mechanism.Slope [▶ 134]. There are two possible settings here:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration;
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = quadratic acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.Start(axLeitachse, Directions.Positive,
Speed1,5000,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Working
Working	√	Working	Working
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

The **state of the group** indicates whether the setpoint generator for all axes in the group is generating a movement or whether the group has been paused.

```
When tecGroup.State.IsEnabled Continue; //The group is in position control mode and is stationary.
When tecGroup.State.IsRunning Continue; //The setpoint generator generates a movement.
```

81.2.2 Reverse()

This function reverses the direction of a currently active, superimposed continuous positioning operation on a secondary axis within a technology group that is coupled via an electric gear. This means that the direction of movement is reversed, and the composite axis moves in the opposite direction at the last set speed.

The following applies:

– The change in direction is performed using the last acceleration slope set.

– If the axis is in the **Enabled** state, the command has no effect.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.Reverse(axLeitachse);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	–		Approved
Working	√	Working	Working
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

81.2.3 NewSpeed()

The function sets a new speed value for the continuous positioning of a composite axis. The following applies:

– The value must be greater than `0`.

– If the axis is already in motion (**Running** state), the new speed value is applied immediately.

  The transition from the current speed to the new speed for a running axis is performed using the values set for the acceleration ramps. This is also the case when the speed decreases.

– If the axis is at a standstill (**Enabled** state), the value is not applied until the next positioning operation.
– The slope type cannot be changed during positioning.
– Jerk limiting is active for all transitions if the slope type is set to *sin ²*.

ⓘ Info

Bring the axis to a stop

To bring an axis to a stop, use only the `MoveHalt` or `Stop` commands. It is **not** possible to stop an axis by setting its speed to `0` .

transfer parameter

Parameters	data type	Meaning	Behavior when the function is called while in the "Running" state
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.	
Speed	Double	Defines the speed at which the axis should be positioned when it is not in the slope (acceleration, deceleration) phases.	If a new positioning operation is initiated while the previous one is still running (Running state), the new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration, and jerk. The speed is adjusted using the acceleration slope parameters.

Syntax

```
tecGroup.MoveVelocity.NewSpeed(axLeitachse,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.
– a state in which calling the function is not recommended, but does not cause an error.
X the state in which calling the function is not permitted and results in an error.

81.2.4 NewAcceleration()

The function sets a new value for the maximum acceleration of a continuous positioning operation on a composite axis.

The following applies:

– An active acceleration slope is not modified.

– The new value will be applied during the next continuous positioning.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start slope dur-ing positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.

Syntax

```
tecGroup.MoveVelocity.NewAcceleration(axLeitachse,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

81.2.5 NewDeceleration()

The function sets a new value for the maximum delay of an infinite positioning cycle for a composite axis.

The following applies:

– The value is also used during an ongoing positioning operation if the target can still be reached using the new parameter.

– Otherwise, the value will be applied during the next positioning.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.

Parameters	data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop slope while positioning. Possible slope types: – *sin²*: The maximum deceleration is reached in the middle of the slope. – *linear*: The slope is driven at maximum deceleration throughout.

Syntax

```
tecGroup.MoveVelocity.NewDeceleration(axLeitachse,2000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

81.2.6 GetDirection(), SetDirection()

Description

Use these functions to read the current direction of a composite axis or to set a new value for that axis. The following applies:

– The value will take effect during the next continuous positioning operation.

– This is also the case if the change is made while positioning is in progress.

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Directions	Directions (optional)	Direction of the continuous motion.
	Directions.Positive	The axis is moved in the positive direction.
	Directions.Negative	The axis is moved in the negative direction.

Syntax

```
AX_Direction := tecGroup.MoveVelocity.GetDirection(axLeitachse);
tecGroup.MoveVelocity.SetDirection(axLeitachse, Directions.Negative);
```

state administration

The function can be read in any state.

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	X	Interrupted
Error: Approved	√	Error: Approved
Error: blocked	√	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

81.2.7 StartSuperPose()

ⓘ Info

APN 51 – Superimposed Positioning

Please refer to the detailed information in **Application Note 051** regarding the superimposed motion of a secondary axis in an electric gear.

This function initiates superimposed continuous positioning of a secondary axis within a technology group that is coupled via an electric gear. The axis begins its movement immediately or applies the data to the current position.

The following applies:

– If the function is executed on a **coupled secondary axis** that is configured as an electric gear, the superim-posed motion is initiated.

– If the function is executed on a **decoupled slave axis**configured as an electric gear or on the **master axis** , that axis initiates a movement that behaves identically to the execution of the Start() [▶ 471] function.

– If the function is executed on a **secondary axis** that is configured as a cam disc, a message is displayed.

transfer parameter

Please follow these rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object; that is, they are properties of the axis. If no value is passed for a parameter, Motion Control uses the last value passed for that parameter.

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.
Directions	Directions (optional) – Directions.Positive – Directions.Negative	Defines the direction of movement (see GetDirection(), SetDirection() [▶ 478]).
Speed	Double (optional)	Defines the speed at which the axis should be positioned when it is not in the slope (acceleration, deceleration) phases. The value must be greater than `0` .

Parameters	data type	Meaning
Acceleration	Double (optional)	Defines the maximum acceleration to be achieved on the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.
Deceleration	Double (optional)	Defines the maximum deceleration to be achieved during the stop ramp for the next `MoveHalt.Stop()` command when positioning, unless another value is specified there. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is driven at maximum deceleration throughout.

Types of ramps

The ramp profiles for the acceleration and deceleration phases are specified by the axis parameter `MCTechno.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (see Slope Type [▶ 136]). There are two possible settings here:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration;
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration;
  deceleration phase = quadratic acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.StartSuperPose(axElGear, Directions.Nega-
tive,200,1000,1000);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked
Approved	√	Working	Approved
Working	√	Working	Approved
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the positioning:

The **state of the group** indicates whether the setpoint generator for all axes in the group is generating a movement or whether the group has been paused.

```
When tecGroup.State.IsEnabled Continue; //The group is in position control
mode and is stationary.
When tecGroup.State.IsRunning Continue; //The setpoint generator generates a
movement.
```

81.2.8 ReverseSuperPose()

This function reverses the direction of a continuous, superimposed positioning operation on a secondary axis within a technology group that is coupled via an electric gear. This means that the direction of movement is reversed, and the composite axis moves in the opposite direction at the last set speed.

The following applies:

– The change in direction is performed using the last acceleration ramp set.

– If the axis is not in an endless positioning mode, the command has no effect.

– If the function is executed on a **decoupled slave axis**configured as an electric gear or on the **master axis** , that axis will begin a movement that behaves identically to the execution of the Reverse() [▶ 184] function.

– A message is displayed when attempting to execute the function on a **secondary axis** that is configured as a cam disc.

ⓘ Info

APN 51 – Superimposed Positioning

Please refer to the detailed information in **Application Note 051** regarding the superimposed motion of a secondary axis in an electric gear.

transfer parameter

Parameters	data type	Meaning
Axis	Pointer to MCAxis	An axis object that must be a member of the technology group and is to be moved.

ⓘ Info

Jerk limit

The sin and sin² ramps are checked to ensure they do not exceed the maximum allowable jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and are adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.ReverseSuperPose(axElGetriebe);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	While running	Final state
Blocked	X		Blocked

Initial state	Allowed	While running	Final state
Approved	–		Approved
Working	√	Working	Working
Interrupted	X		Interrupted
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

82 MCTechno.Power

The **MCTechno.Power** function allows you to enable or disable the control—and thus the torque generation—of a technology group. All axes involved in the group are switched to the new operating state simultaneously.

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

82.1 Syntax for MCTechno.Power

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Power.<Function>(optional <Parameter>;
```

82.2 Features of MCTechno.Power

This function offers the following functions:

– Enable()
– Disable()
– Quickstop()

82.2.1 Enable()

This function activates the controllers of the servo amplifiers for all coupled axes and applies power to the mo-tor. A technology group can only be activated if all group axes are operational.

transfer parameter

The function has no transfer parameters.

Syntax

```
tecGroup.Power.Enable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Approved
Approved	–	Approved
Working	–	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Therefore, check whether the new state has been reached in the program before calling any further commands for the group.

```
If tecGroup.State.IsEnabled , continue;
```

82.2.2 Disable()

This function disables the servo amplifier on all group axes for torque generation (pulse inhibit). The motor will immediately lose power and coast to a stop if no brake is applied.

transfer parameter

Parameters	data type	Meaning
DisableModes	MCPowerDisableMode (optional)	Mode for executing the `Disable()` function.
	MCPowerDisableMode.Forced (Default)	The pulse block takes effect immediately in all cases. This mode is also active if no parameter is specified.
	MCPowerDisableMode.StateCheck	The pulse inhibit function is only active when the group is in the **"Enabled"**operating state. In all other states, an error message is displayed.

Syntax

```
tecGroup.Power.Disable();
tecGroup.Power.Disable(MCPowerDisableModes.Forced);
tecGroup.Power.Disable(MCPowerDisableModes.StateCheck);
```

state administration

The following table applies when writing the function:

The following table applies when the function is called without parameters or with the parameter `` `MCPowerDisableModes.Forced` ``:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	√	Blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The following table applies to calls using the `MCPowerDisableModes.StateCheck` parameter:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before executing any further commands for the group, check whether the new state has been reached in the program as follows:

```
If tecGroup.State.IsDisabled , continue;
```

If the axis was previously in the **"Error: Enabled"** state, use the following command.

```
When geoGroup.State.IsDisabledError Continue;
```

82.2.3 QuickStop()

This function stops all coupled axes using the quick-stop deceleration and locks the servo amplifiers once a standstill is reached.

NOTICE



No position control of the coupled axes during a quick stop

During the rapid stop delay, no position control takes place, and therefore there is no coordinated movement of the coupled axes as defined by a cam disc or an electric gear. Each coupled axis comes to a stop at a different time, depending on its current speed and the quick-stop deceleration setting.

► Use this command only in emergencies!

transfer parameter

The function has no transfer parameters.

Syntax

```
tecGroup.Power.QuickStop();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	–	Blocked
Approved	√	Blocked
Working	√	Blocked
Interrupted	√	Blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

The state transition cannot be processed within a single MCX cycle. Before executing any further commands for the group, check whether the new state has been reached in the program as follows:

```
If tecGroup.State.IsDisabled , continue;
```

82.3 Example Task: MCTechno.Power

Here is an example task for a single axis that includes various Motion API functions:

```
Task Sample 1 : Autorun
    //Unlock the coupled axes
    If tecGroup.State.IsDisabled Then
        tecGroup.Power.Enable();
    End_if;
    If tecGroup.State.IsEnabled , continue;

    //A process may be associated with the group here, or other tasks may
move the group
    //…

    // Lock the group
    If tecGroup.State.IsRunning Then    //if the group is still moving, stop
after a reasonable delay

        tecGroup.MoveHalt.Start(Techno.Members.Master,
MCTechnoHaltModes.Normal,1000.0);
    When tecGroup.Mechanism.Slope.IsStopped , continue;

    End_if;
    tecGroup.Power.Disable(); // Disable the group
    If tecGroup.State.IsDisabled , continue;

End_Task;
```

83 MCTechno.Settings

The **MCTechno.Settings** function allows you to adjust or save all parameters for an MCX technology group at once.

Do not adjust parameters while a save operation is in progress

Saving the parameters to a file is a process that runs in the background. In other words, it spans several cycles of the MCX. If parameter values are changed while the data is being saved, it is impossible to predict whether the old or new value will be written to the file.

How Parameter Management Works

Parameters saved in JetSym 's motion setup are stored in a text file on the PC and, if requested, also in the controller's file system (Motionsubfolder. The filename of the parameter file on the controller is *MCTechno_X.ini*, where *X* stands for the technology group ID number. The technology group ID is displayed on the Properties page in the MCX group's motion setup. In addition, the tooltip in the Hardware tree displays the technology group ID when the mouse pointer hovers over the group.

When the controller boots up, the MCX kernel reads the data from the *MCTechno_X.ini*file and uses it for the relevant group. If such a file is present on the controller, no further parameterization is required. If no corresponding file exists for the group, the parameters are loaded with default values.

Normally, no modifications to the file are required from within the STX program.

In very rare cases, however, it may be advisable to reload the parameter file onto the controller or to save the parameters. However, when saving, it is important to note that the parameter file saved in the JetSym project may no longer match the parameters currently in effect on the machine.

83.1 Syntax for MCTechno.Settings

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Settings.<Function>(optional <Parameter>;
```

83.2 Features of MCTechno.Settings

This function offers the following functions:

– SaveToFile()
– LoadFromFile()

83.2.1 SaveToFile()

transfer parameter

Parameters	data type	Meaning
Suffix	Integer (optional)	Enables saving the parameters of the MCX group to a file in the "Motion" subfolder within the controller's file system. Depending on the transfer parameter, different files are created or overwritten: – Parameter equal to `0` or no parameter: The *MCTechno_X.ini* parameter file for this group is created or overwritten with the current values. – Parameter not equal to `0`: The parameter file *MCTechno_X_TmpY.ini* is created or overwritten with the current values. In this case , *Y* is equal to the parameter `suffix`.

Syntax

```
tecGroup.Settings.SaveToFile();    // Overwrites (or creates) the file
MCTechno_1.ini (if tecGroup has Techno ID 1)
tecGroup.Settings.SaveToFile(3);    // Creates the file MCTecho_1_Tmp3.ini
(if tecGroup has Techno ID 1)
```

state administration

The function can be called from any state.

83.2.2 LoadFromFile()

transfer parameter

Parameters	data type	Description
Suffix	Integer (optional)	Enables the import of MCX group parameters from a file located in the "Motion" subfolder within the controller's file system. Depending on the transfer parameter, different files are read. – Parameter equal to `0` or no parameter: The data from the *MCTechno_X.ini* file is read and used for the relevant group. – Parameter not equal to `0`: The data from the *MCTechno_TmpY.ini* file is read and used for the relevant group. In this case , *Y* is equal to the parameter `suffix`.

Syntax

```
tecGroup.Settings.LoadFromFile();    // Reads the MCTechno_1.ini file
(if tecGroup has Geo-ID 1)
tecGroup.Settings.LoadFromFile(3);    // Reads the file
MCTechno_1_Tmp3.ini (if tecGroup has Geo-ID 1)
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	X	Approved
Working	X	Working
Interrupted	X	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

84 MCTechno.State

The **MCTechno.State** function allows you to retrieve various details about a technology group, such as operating state, motion type, and so on.

Basic and additional information

In this context, please also note the following information:

– For state transitions, see the MCTechno.State.Transitions [▶ 498]section;
– For a detailed description of the MCX core architecture, see the JetSymonline help
– For basic information, see Motion Control Application Note No. 54 (MCX Motion Control Basics).

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

Operating status

At any given time, a group is in one, and only one, clearly defined operating state (`OperatingState`). For example, if the group is in the **"Running"** operating state, the **"Enabled"** operating state is not active, even though all group axes naturally have pulse enable.

In a given state, only a subset of all function calls is permitted. The state model for the axis is as follows:

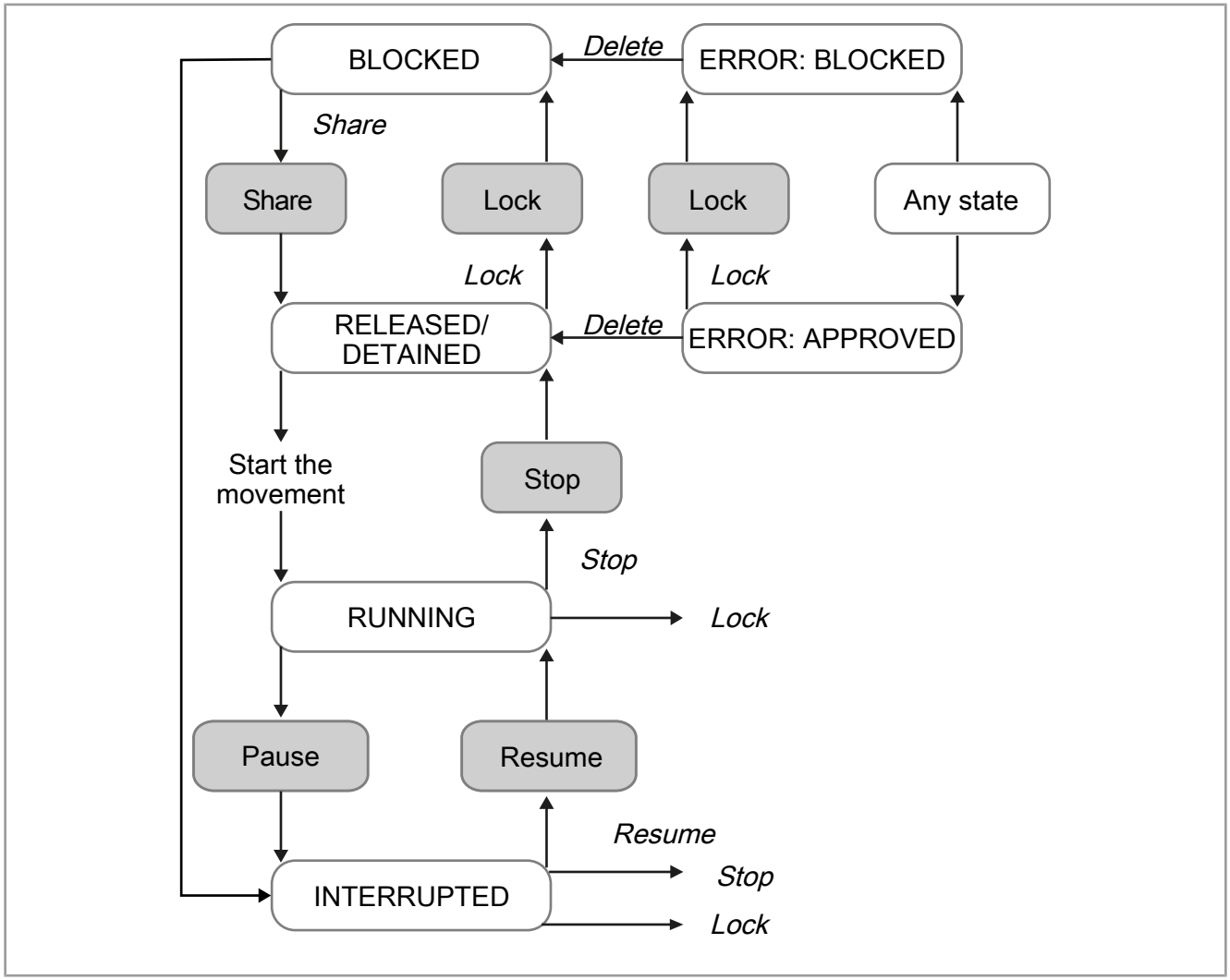


Fig. 15: State-space model for MCX groups

Operating status	Meaning
Inactive	The group cannot accept commands, for example, because it has not been activated.
Disabled	– The servo amplifier controllers for all coupled axes are open (no pulse enable). – No torque is generated.
Enabled	– The controllers for all coupled axes are closed (pulse enable). – The setpoint interpolator is at a standstill.
Running (In progress)	At least one axis is in motion; that is, the setpoint interpolator transmits motion setpoints to the servo amplifier.
Suspended	– This corresponds to the **"Enabled"** operating state, with the exception that the target position is retained. A motion that has been paused using `MCTechno.State.Transitions.Suspend()` can be resumed using the `MCTechno.State.Transitions.Resume()` command.
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control mode (error level 4).
DisabledError (Error: Locked)	An error has occurred. After the error response, control of all axes is disabled (error levels 5 and 6).

Tab. 10: Operating states of technology groups

operating state

The operating state (`OperatingMode`) specifies the general way in which the technology group operates. There are 3 operating states:

operating state	Meaning
`Inactive`	The group cannot accept commands, for example, because it has not been activated.
`Normal` (normal)	The group is usable. Traverse commands are always executed on a first-in, first-out basis, since buffered operation is not possible with technology groups.
`Error`	An operator error has occurred. The operating state is retained; however, no new functions or parameters can be passed.

Tab. 11: Operating States of MCX Technology Groups

Type of movement

The motion type (`TypeOfMotion`) specifies the type of motion that the setpoint generator is currently generating for the technology group.

Querying the transaction type is useful, for example, in a monitoring task where the process is unknown. Based on the active motion type prior to shutdown, it is possible to control the restart correctly.

Operating status	Meaning
No movement (`MoveNone`)	The setpoint generator is out of service because no movement is requested.
Positioning (`MovePtp`)	The leading axis of the technology group is currently, or was most recently, moving from point to point. This transaction type remains active even after the group has been activated for the first time. The motion type of a secondary axis that is started independently of the master axis is not taken into account.
Continuous motion (`MoveVelocity`)	The index of the technology group is currently, or was most recently, in a continuous trend. The motion type of a secondary axis that is started independently of the master axis is not taken into account.

84.1 Syntax for MCTechno.State

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.State.<Access to Value>
```

84.2 Properties of MCTechno.State

This function offers the following properties:

- Value
- IsDisabled
- IsEnabled
- IsRunning
- IsSuspended
- IsDisabledError
- IsEnabledError
- IsInactive

– Operating Mode
– TypeOfMotion
– Transitions (for a description, see MCTechno.State.Transitions [▶ 498])

84.2.1 Value

Description

data type	Meaning
Operating States	Reads the operational status of the technology group (for a detailed description, see the *"Operational Status"* section in MCGeo.State.Transitions [▶ 387]).
OperatingStates.Inactive	Inactive
OperatingStates.Disabled	Blocked
OperatingStates.Enabled	Approved
OperatingStates.Running	Working
OperatingStates.Suspended	Interrupted
OperatingStates.EnabledError	Error: Approved
OperatingStates.DisabledError	Error: blocked

Syntax

```
OpState := tecGroup.State.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.2 IsDisabled

Description

data type	Meaning
Bool	Check whether the group is in the **"Locked"** state.

Syntax

```
When techgroup.State.IsDisabled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.3 IsDisabledError

Description

data type	Meaning
Bool	Check whether the group is in the **"Error: Locked"** state.

Syntax

```
When tecGroup.State.IsDisabledError , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.4 IsEnabled

Description

data type	Meaning
Bool	Checks whether the group is in the **"Enabled"** state.

Syntax

```
If tecGroup.State.IsEnabled , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.5 IsEnabledError

Description

data type	Meaning
Bool	Check whether the group is in the **"Error: Enabled"** operational state.

Syntax

```
When tecGroup.State.IsEnabledError , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.6 IsInactive

Description

data type	Meaning
Bool	Check whether the group is in the **inactive** state.

Syntax

```
If tecGroup.State.IsInactive , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.7 IsRunning

Description

data type	Meaning
Bool	Check whether the group is in the **"Running"** state.

Syntax

```
If tecGroup.State.IsRunning , Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.8 IsSuspended

Description

data type	Meaning
Bool	Check whether the group is in the **"Disconnected"** state.

Syntax

```
If tecGroup.State.IsSuspended , Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.9 OperatingMode.Value

Description

data type	Meaning
Type MCOperatingModes	Reads the operating state of the group (for a detailed description, see the "Operating State" [▶ 387]section).
MCOperatingModes.Inactive	The group is inactive, for example, because it has not been activated.
MCOperatingModes.Normal	The group has been activated and is not in an error state.
MCOperatingModes.Error	The group is in an error state due to an operator error.

Syntax

```
OpModeAxis := tecGroup.State.OperatingMode.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInInactiveMode

Description

data type	Meaning
Bool	Check whether the axis is in the *inactive* operating state.

Syntax

```
When tecGroup.State.OperatingMode.IsInInactiveMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInErrorMode

Description

data type	Meaning
Bool	Check whether the group is in the *error* operating state.

Syntax

```
If tecGroup.State.OperatingMode.IsInErrorMode , then continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

OperatingMode.IsInNormalMode

Description

data type	Meaning
Bool	Check whether the group is in the *normal* operating state.

Syntax

```
When tecGroup.State.OperatingMode.IsInNormalMode , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

84.2.10 TypeOfMotion.Value

Description

Operating status	Meaning
Type MCTechnoTypesOfMotion	The current state of the technology group (for a detailed description, see the section MCTechno.State [▶ 491]).

Operating status	Meaning
MCTechnoTypesOfMotion.MoveNone	Movement type : `None`: The group is not active.
MCGeoTypesOfMotion.MovePtp	The `"Positioning"` movement type remains displayed after the technology group is activated until another movement type is performed.
MCGeoTypesOfMotion.MoveVelocity	The `"Endless"` transaction type remains displayed until the technology group executes a different transaction type.

Syntax

```
TypeOfMotion := tecGroup.State. TypeOfMotion.Value;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveNone

Description

data type	Meaning
Bool	Check whether the group is in the `"None"` transaction type.

Syntax

```
When tecGroup.State.TypeOfMotion.IsMoveLinear , continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMovePtp

Description

data type	Meaning
Bool	Checks whether the group is currently performing a `positioning operation .`

Syntax

```
When tecGroup.State.TypeOfMotion.IsMovePtp Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

TypeOfMotion.IsMoveVelocity

Description

data type	Meaning
Bool	Check whether the group is currently performing a `continuous motion`.

Syntax

```
When tecGroup.State.TypeOfMotion.IsMoveVelocity Continue;
```

state administration

This property is used for diagnostic purposes. They can only be accessed by reading them.

The property can be read in any state.

85 MCTechno.State.Transitions

The **MCTechno.State.Transitions** function allows you to control state transitions in a path group for specific applications.

As a rule, state transitions in a group are triggered automatically by motion commands and the behavior of the axes. Direct control of state administration is required only in exceptional cases—for example, to respond to an error. Using the **MCTechno.State.Transitions**function, as well as `Diagnostics.ClearErrors()`, state transitions can be explicitly triggered.

The **MCTechno.State** function allows you to retrieve various pieces of information about a technology group, such as operating state, motion type, etc.

Basic and advanced information

In this context, please also note the following information:

– For a description of the other functions of MCTechno.State, see the MCTechno.State [▶ 489]section;
– For a detailed description of the MCX core architecture, see the JetSy monline help
– For basic information, see Motion Control Application Note No. 54 (MCX Motion Control Basics).

ⓘ Info

Further information

Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

Operating status

At any given time, a group is in one, and only one, clearly defined operating state (`OperatingState`). For example, if the group is in the **"Running"** operating state, the **"Enabled"** operating state is not active, even though all group axes naturally have pulse enable.

In a given state, only a subset of all function calls is permitted. The state model for the axis is as follows:

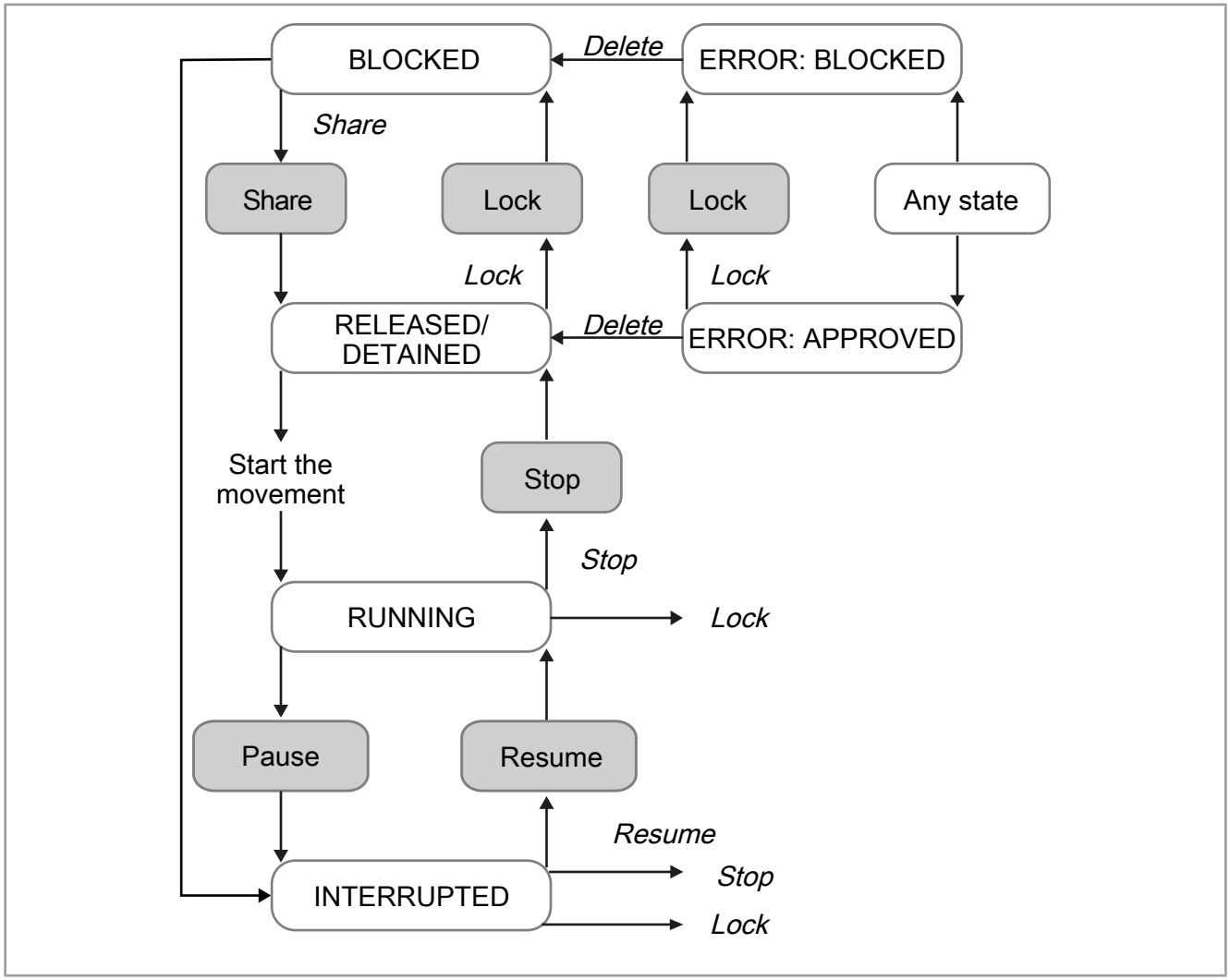


Fig. 16: State-space model for MCX groups

Operating status	Meaning
Inactive	The group cannot accept commands, for example, because it has not been activated.
Disabled	– The servo amplifier controllers for all coupled axes are open (no pulse enable). – No torque is generated.
Enabled	– The controllers for all coupled axes are closed (pulse enable). – The setpoint interpolator is at a standstill.
Running (In progress)	At least one axis is in motion; that is, the setpoint interpolator transmits motion setpoints to the servo amplifier.
Suspended	– This corresponds to the **"Enabled"** operating state, with the exception that the target position is retained. A motion that has been paused using `MCTechno.State.Transitions.Suspend()` can be resumed using the `MCTechno.State.Transitions.Resume()` command.
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control mode (error level 4).
DisabledError (Error: Locked)	An error has occurred. After the error response, control of all axes is disabled (error levels 5 and 6).

Tab. 12: Operating states of technology groups

operating state

The operating state (`OperatingMode`) specifies the general way in which the technology group operates. There are 3 operating states:

operating state	Meaning
`Inactive`	The group cannot accept commands, for example, because it has not been activated.
`Normal` (normal)	The group is usable. Traverse commands are always executed on a first-in, first-out basis, since buffered operation is not possible with technology groups.
`Error`	An operator error has occurred. The operating state is retained; however, no new functions or parameters can be passed.

Tab. 13: Operating States of MCX Technology Groups

Type of movement

The motion type (`TypeOfMotion`) specifies the type of motion that the setpoint generator is currently generating for the technology group.

Querying the transaction type is useful, for example, in a monitoring task where the process is unknown. Based on the active motion type prior to shutdown, it is possible to control the restart correctly.

Operating status	Meaning
No movement (`MoveNone`)	The setpoint generator is out of service because no movement is requested.
Positioning (`MovePtp`)	The leading axis of the technology group is currently, or was most recently, moving from point to point. This transaction type remains active even after the group has been activated for the first time. The motion type of a secondary axis that is started independently of the master axis is not taken into account.
Continuous motion (`MoveVelocity`)	The index of the technology group is currently, or was most recently, in a continuous trend. The motion type of a secondary axis that is started independently of the master axis is not taken into account.

85.1 Syntax for MCTechno.State.Transitions

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.State.Transitions.<Function>(optional <Parameter>);
```

85.2 Functions of MCTechno.State.Transitions

This function offers the following functions:

- Enable()
- Disable()
- Stop()
- ErrorDisable()
- ErrorStop()
- ErrorLockUserInput()
- Suspend()
- Resume()

85.2.1 Enable()

The `State.Transitions.Enable()` function is identical to the `MCTechno.Power.Enable()` function. For a detailed description of the function, see the MCTechno.Power.Enable() [▶ 483]section.

Syntax

```
tecGroup.State.Transitions.Enable();
```

85.2.2 Disable()

Depending on the parameter passed (see the table below), this function performs the same action as the `MCTechno.Power.Disable()` and `MCTechno.Power.Quickstop()` commands. For more details, please refer to the descriptions for MCTechno.Power [▶ 483].

transfer parameter

Parameters	data type	Meaning
Mode	MCStateTransitionDisableModes	Mode for executing the `Disable()` function
	MCStateTransitionDisable-Modes.ImmediatePulseBlocking	Performs the same function as the `MCTechno.Power.Disable()` command or `MCTechno.Power.Disable(MCPowerDisable-Modes.Forced)`
	MCStateTransitionDisable-Modes.VelocityControlSlope	Performs the same function as the `MCTechno.Power.QuickStop()` command.

Syntax

```
tecGroup.State.Transitions.Disable(MCStateTransitionDisableModes.Immedi-
atePulseBlocking);
tecGroup.State.Transitions.Disable(MCStateTransitionDisableModes.VelocityCon
trolSlope);
```

85.2.3 Stop()

Description

Depending on the parameter passed (see table below), this function performs the same action as the MC-Geo.Stop() [▶ 399]command.

The difference is that passing a function parameter is mandatory when using `MCGeo.State.Transi-tion.Stop()` , whereas it is optional when using `MCGeo.Stop()` (see there for details).

transfer parameter

Parame-ters	data type	Meaning
Mode	MCStateTransitionStop-Modes	There are two delay values available.
	StopModes.ProgDecel-eration	The deceleration value last specified for the group is used for stop-ping.
	StopModes.MaxDeceler-ation	To bring the system to a stop, the maximum acceleration of the group is used as deceleration. This was either specified during commissioning in the motion setup under "Maximum Deceleration" or programmed using the command `geoGroup.Mechanism.Limitations.MaxAcceleration` .

Syntax

```
geoGroup.State.Transitions.Stop(MCStateTransitionStopModes.ProgDecelera-
tion);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	X		Blocked
Approved	√		Approved
Working	√	Working	Approved
Interrupted	√		Approved
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Condition Monitoring

Various pieces of information are available to help you monitor the positioning.

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis is stopped.

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and is stationary.
When geoGroup.State.IsRunning Continue; //The setpoint generator for at
least one joint axis is generating motion.
```

– The **ramp status** indicates which phase of the motion cycle the setpoint generator is currently in.

```
When geoGroup.Mechanism.Slope.IsAccelerating Continue;
When geoGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When geoGroup.Mechanism.Slope.IsDecelerating Continue;
When geoGroup.Mechanism.Slope.IsStopped , continue;
```

– This function does not affect the **target window status** .

85.2.4 ErrorDisable()

This function triggers an error in the path group, resulting in the **"Error: Locked"** state. The error response can be configured (see the `ErrorDisableMode`property).

transfer parameter

The function has no transfer parameters.

Syntax

```
tecGroup.State.Transitions.ErrorDisable();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: blocked
Approved	√	Error: blocked
Working	√	Error: blocked
Interrupted	√	Error: blocked
Error: Approved	√	Error: blocked
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

85.2.5 ErrorStop()

This function triggers an error in the group, resulting in the **"Error: Released"** state. The error response can be configured (see the ErrorStopMode [▶ 507]property).

transfer parameter

The function has no transfer parameters.

Syntax

```
tecGroup.State.Transitions.ErrorStop();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Error: Approved
Approved	√	Error: Approved
Working	√	Error: Approved
Interrupted	√	Error: Approved
Error: Approved	-	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

85.2.6 ErrorLockUserInput()

This function triggers an operational error in the group.

transfer parameter

The function has no transfer parameters.

Syntax

```
tecGroup.State.Transitions.ErrorLockUserInput();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	√	Blocked
Approved	√	Approved
Working	√	Working
Interrupted	√	Interrupted
Error: Approved	–	Error: Approved
Error: blocked	–	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

85.2.7 Suspend()

Use this function to interrupt a group movement that is currently in progress. Deceleration occurs either using the last programmed deceleration ramp or the configured maximum deceleration. The state after stopping is not **"Released"** but **"Paused"**. The `Resume()` function allows the interrupted path movement of a group to be resumed and moved to the original target position (the reference axis).

If the reference axis cannot be stopped before the actual target position within the specified deceleration time, the command is ignored and the positioning is completed. The state is then **"Released"** and not **"Interrupted"**.

transfer parameter

Parameters	data type	Meaning
Mode	MCStateTransitionStopModes	Mode for the deceleration ramp of the `Suspend()` function
	MCStateTransitionStopModes.ProgrammedDeceleration	Uses the last deceleration value programmed for the reference axis.
	MCStateTransitionStopModes.MaxDeceleration	Uses the maximum deceleration configured for the reference axis.

Syntax

```
tecGroup.State.Transitions.Suspend(MCStateTransitionStopModes.ProgDeceleration);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	X	Approved
Working	√	Interrupted
Interrupted	–	Interrupted
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

85.2.8 Resume()

This function resumes positioning that was interrupted by the Suspend() [▶ 504] function. The system moves to the originally specified target position of the leading axis. The speed and acceleration values are also carried over from the original position.

The `Resume()` function only takes effect if the group is in the **"Paused"** state.

transfer parameter

The function has no transfer parameters.

Syntax

```
tecGroup.State.Transitions.Resume();
```

state administration

The following table applies when writing the function:

Initial state	Allowed	Final state
Blocked	X	Blocked
Approved	X	Approved
Working	–	Working
Interrupted	√	Working
Error: Approved	X	Error: Approved
Error: blocked	X	Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

85.3 Properties of MCTechno.State.Transitions

This function offers the following properties:

– ErrorDisableMode
– ErrorStopMode
– ErrorLevel

85.3.1 ErrorDisableMode

Description

data type	Meaning
MCStateTransitionDisableModes	Error response used when the `ErrorDisable()` function is called.
MCStateTransitionDisableModes.Velocity-ControlSlope	Stops the group using the emergency brake. Once a standstill is reached, the servo amplifier is disabled.
MCStateTransitionDisableModes.Immedi-atePulseBlocking	Immediately disables the servo amplifier from generating torque (pulse inhibit).

Syntax

```
tecGroup.State.Transitions.ErrorDisableMode := MCStateTransitionDisable-
Modes.ImmediatePulseBlocking;
Error response := tecGroup.State.Transitions.ErrorDisableMode;
```

state administration

The property can be read and written in any state.

When the property is read, the currently valid error response for the command is returned.

85.3.2 ErrorStopMode

Description

data type	Meaning
MCStateTransitionErrorStopModes	Error response used when the `ErrorStop()` function is called.
MCStateTransitionErrorStop-Modes.ProgrammedDeceleration	Pauses the group for the programmed delay.
MCStateTransitionErrorStop-Modes.MaxDeceleration	Pauses the group for the maximum delay.
MCStateTransitionErrorStop-Modes.AtBlockEnd	This mode is irrelevant for the technology group, as buffered operation is not possible with the technology group.

Syntax

```
tecGroup.State.Transitions.ErrorStopMode := MCStateTransitionStop-
Modes.MaxDeceleration;
Error response := tecGroup.State.Transitions.ErrorStopMode;
```

state administration

The property can be read and written in any state.

85.3.3 ErrorLevel

Description

Use this property to set the error level that the path group adopts in the event of a drive error. By default, this is error level 4 (Operation STOP and **error** status **: enabled**).

Data type	Meaning
MCStateTransitionErrorLevels	Error level of the path group in the event of a drive error. For detailed information on the differences between error response levels, see the " *Error Management"*section in the JetSym online help.
MCStateTransitionErrorLevels.Message	The group responds with error level 1: A notification/alert is generated.
MCStateTransitionErrorLevels.Channel-Stop	The group responds with error level 2: The command channel is being blocked.
MCStateTransitionErrorLevels.Channel-StopClearBuffer	Exception for technology groups: This function is identical to that of error response level 2, since buffered operation is not possible with technology groups.
MCStateTransitionErrorLevels.Stop	The group responds with error level 4: The "Stop" command is executed (default).
MCStateTransitionErrorLevels.Quickstop (Default)	The group responds with error level 5: An emergency stop is initiated.
MCStateTransitionErrorLevels.PulseBlock-ing	The group responds with error level 6: The controller is disabled (pulse disable).

Syntax

```
tecGroup.State.Transitions.ErrorLevel := MCStateTransitionErrorLevels.Pulse-
Blocking;
ReactionOnDriveError := tecGroup.State.Transitions.ErrorLevel;
```

state administration

The property can be read and written in any state.

86 MCTechno.Stop

The **MCTechno.Stop** function allows you to bring a moving technology group to a stop using a defined ramp.

Unlike the `MoveHalt.Start()` function, the `Stop`function always affects the entire group.

This function has no additional functions or properties.

> **ⓘ Info**
>
> **Further information**
>
> Please refer to **Motion Control Application Notes (APN) Nos. 49 – 56** for procedures and sample projects related to working with groups. These are available for download on the Bucher Automation website.

86.1 Syntax for MCTechno.Stop

The easiest way to enter the function in the JetSym Editor is as follows, using IntelliSense:

```
<Axis group name>.Stop(optional <Parameter 1>)
```

86.2 Features of MCTechno.Stop

`MCTechno.Stop` is a function of the group object.

86.2.1 Stop()

transfer parameter

Please note the following guidelines:

- This function executes the stop command immediately. The parameter passed (see table) specifies the maximum delay for this.
- If no parameter is passed, the last `StopMode` passed is used.
- When the system stops, the secondary axes come to a halt in sync with the master axis, in accordance with their coupling function.

Parameters	data type	Meaning
StopModes	Enum StopModes (optional)	There are two delay values available.
	StopModes.ProgDeceleration	The deceleration value last specified for the group is used for stopping (default).
	StopModes.MaxDeceleration	To stop, the maximum acceleration of the lead axis is used as deceleration. This was either specified during commissioning in the Motion Setup under “Maximum Deceleration” or programmed using the command `axLeitachse.Mechanism.Limitations.MaxAcceleration`.

Syntax

```
tecGroup.Stop(StopModes.ProgDeceleration);
tecGroup.Stop(StopModes.MaxDeceleration);
```

state administration

The following table applies when writing the function:

Initial state	Allowed	During execution	Final state
Blocked	X		Blocked
Approved	√		Approved
Working	√	Working	Approved
Interrupted	√		Approved
Error: Approved	X		Error: Approved
Error: blocked	X		Error: blocked

The fact is:

√ the state in which the function can be called.

– a state in which calling the function is not recommended, but does not cause an error.

X the state in which calling the function is not permitted and results in an error.

Condition Monitoring

Various pieces of information are available to help you monitor the `Stop()` function.

– **The axis state** indicates whether the setpoint generator is generating a movement or whether the axis is stopped.

```
When tecGroup.State.IsEnabled Continue; //The group is in position control
mode and all axes are at rest.
When tecGroup.State.IsRunning Continue; //The setpoint generator for at
least one coupled axis is generating motion.
```

– The **ramp status** indicates which phase of the motion cycle the setpoint generator is currently in.

```
When tecGroup.Mechanism.Slope.IsAccelerating Continue;
When tecGroup.Mechanism.Slope.IsAtConstantSpeed Continue;
When tecGroup.Mechanism.Slope.IsDecelerating , continue;
When tecGroup.Mechanism.Slope.IsStopped , continue;
```

– This function does not affect the **target window status** .

Glossary

Error
A notification type that indicates that an error threshold has been reached. Errors trigger a system response and must be acknowledged before operation can continue. See also "Warnings" and "Messages."

Master axis
In the context of the MCCoupling Group function, the master axis is a superimposed axis that should not be confused with the reference axis, as their functions are fundamentally different.

synchronous axis
Axes that move in sync with the path movement without being mechanically connected to it.

technology group
A technology group is a coupling of multiple MCX axes in which one or more slave axes are moved based on the position of a master axis.

Warning
A notification type for EtherCAT devices that warns before an error threshold is reached. Warnings do not trigger any system response and are automatically deleted as soon as the cause of the warning no longer exists. See related reports.

Bucher Automation AG
Thomas-Alva-Edison-Ring 10
71672 Marbach/Neckar, Germany
T +49 7141 2550-0
info@bucherautomation.com

www.bucherautomation.com