BUCHER automation

Motion-API-Documentation



60888951_01

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	1.01
Date of issue	11/16/2026

Table of Contents

1 Introduction ... 26
1.1 Information about this document ... 26
1.2 Typographical conventions ... 26
2 MCAxis.AxisConfigData ... 27
2.1 Syntax for MCAxis.AxisConfigData ... 27
2.2 Properties of MCAxis.AxisConfigData ... 27
2.2.1 DeviceName ... 27
2.2.2 AxisMode ... 28
2.2.3 AxisDriveType ... 28
2.2.4 ObjectID ... 29
2.2.5 DeviceID ... 29
2.2.6 IsSimulation ... 29
3 MCAxis.Buffer ... 31
3.1 Syntax for MCAxis.Buffer ... 31
3.2 Functions of MCAxis.Buffer ... 31
3.2.1 Create() ... 32
3.2.2 Clear() ... 32
3.2.3 Delete() ... 33
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 Functions 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 ... 42

4.3.4 MessageCount ... 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 ActualDCLinkVoltage ... 45
5.2.2 ActualHeatSinkTemperature ... 46
5.2.3 ActualInternalTemperature ... 46
5.2.4 DCLinkVoltageMax ... 46
5.2.5 DCLinkVoltageMin ... 47
5.2.6 PwmFrequency ... 47
5.2.7 PwmFrequencyReduction ... 47
5.2.8 RatedCurrent ... 48
5.2.9 RatedDCLinkVoltage ... 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 Functions 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 ActualMotorCurrent ... 54
7.2.4 ActualMotorVoltage ... 54
7.2.5 MaxMotorCurrent ... 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 OverloadFactor ... 63
9.2.9 MaxMotorSpeed ... 63
9.2.10 SetpointLimit ... 64
9.2.11 DifferenceWindow ... 64
9.2.12 TotalInertia ... 64
9.2.13 TorqueFeedForwardScale ... 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 OverloadFactor ... 67
10.2.4 MaxMotorTorque ... 67
10.2.5 SetpointLimit ... 68
11 MCAxis.Drive.Diagnostics ... 69
11.1 Syntax for MCAxis.Drive.Diagnostics ... 69
11.2 Functions 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 PolarityReferenceSwitch ... 72
12.2.2 LimitSwitchMode ... 73
12.2.3 PolarityLimitSwitchMin ... 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 GearEncoder ... 77
13.2.5 MultiTurnPosition ... 78
13.2.6 SingleTurnPosition ... 78
13.2.7 MaxNoOfAbsRevolutions ... 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.OSVersion ... 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 IsReadyForOperation ... 85
17 MCAxis.Drive.Motor ... 86
17.1 Syntax for MCAxis.Drive.Motor ... 86
17.2 Functions of MCAxis.Drive.Motor ... 86
17.2.1 MeasureCommutationOffset() ... 86
17.3 Properties of MCAxis.Drive.Motor ... 87
17.3.1 MotorDesign ... 88
17.3.2 RatedCurrent ... 89

17.3.3 MaxCurrent ... 90
17.3.4 RatedSpeed ... 90
17.3.5 RatedTorque ... 91
17.3.6 Inductance ... 91
17.3.7 Resistance ... 92
17.3.8 VoltageConstant ... 93
17.3.9 TorqueConstant ... 93
17.3.10 PolePairs ... 94
17.3.11 CommutationOffset ... 95
17.3.12 ThermalDelayTime ... 95
17.3.13 ShaftPosition ... 96
17.3.14 Inertia ... 96
17.3.15 TemperatureSensorType ... 97
17.3.16 TemperatureSensorConnector ... 98
17.3.17 Temperature ... 99
17.3.18 I2tOperatingMode ... 99
17.3.19 I2tValue ... 101

18 MCAxis.Drive.QuickStopDeceleration ... 102

18.1 Syntax for MCAxis.Drive.QuickStopDeceleration ... 102
18.2 Properties of MCAxis.Drive.QuickStopDeceleration ... 102
18.2.1 QuickStopDeceleration ... 102

19 MCAxis.Drive.TorqueShutOff ... 103

19.1 Syntax for MCAxis.Drive.TorqueShutOff ... 105
19.2 Functions of MCAxis.Drive.TorqueShutOff ... 105
19.2.1 Configure() ... 105
19.2.2 ReadConfiguration() ... 106
19.2.3 Activate() ... 107
19.2.4 Deactivate() ... 108
19.3 Properties of MCAxis.Drive.TorqueShutOff ... 108
19.3.1 State ... 108
19.4 Example task MCAxis.Drive.TorqueShutOff ... 108

20 MCAxis.Info ... 110

20.1 Syntax for MCAxis.Info ... 110
20.2 Properties of the functionality ... 110
20.2.1 Name ... 110
20.2.2 ObjectID ... 110
20.2.3 ClassID ... 111
20.3 Example task MCAxis.Info ... 111

21 MCAxis.IsInTargetWindow 112
21.1 Syntax for MCAxis.IsInTargetWindow 112
21.2 Properties of MCAxis.IsInTargetWindow 112
21.2.1 IsInTargetWindow 112
21.3 Example task MCAxis.IsInTargetWindow 112
22 MCAxis.Mechanism.AxisMode 114
22.1 Syntax for MCAxis.Mechanism.AxisMode 115
22.2 Properties of MCAxis.Mechanism.AxisMode 115
22.2.1 AxisMode 115
23 MCAxis.Mechanism.AxisType 116
23.1 Syntax for MCAxis.Mechanism.AxisType 116
23.2 Properties of MCAxis.Mechanism.AxisType 116
23.2.1 AxisType 116
23.3 Description of the axis types 117
23.3.1 Rotary axis 117
23.3.2 Linear axis 118
23.3.3 Non-linear eccentric axis 119
23.3.4 Non-linear piston axis 119
23.4 Example task for MCAxis.Mechanism.AxisType 120
24 MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor 121
24.1 Syntax for MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor 121
24.2 Properties of MCAxis.Mechanism.GearRatioLoad, MCAxis.Mechanism.GearRatioMotor 121
24.2.1 GearRatioMotor 121
24.2.2 GearRatioLoad 122
25 MCAxis.Mechanism.Kinematics 123
25.1 Syntax for MCAxis.Mechanism.Kinematics 123
25.2 Functions of MCAxis.Mechanism.Kinematics 123
25.2.1 Crank.Configure(), Piston.Configure() 123
25.3 Properties of MCAxis.Mechanism.Kinematics 124
25.3.1 .Offset / .InvertDirection / .Radius / .RodLength 124
25.4 Example task MCAxis.Mechanism.Kinematics 125
26 MCAxis.Mechanism.Limitations 126
26.1 Syntax for MCAxis.Mechanism.Limitations 126
26.2 Functions of MCAxis.Mechanism.Limitations 126
26.2.1 SetTravelLimits() 126
26.3 Properties of MCAxis.Mechanism.Limitations 127
26.3.1 TravelLimitMin 127
26.3.2 TravelLimitMax 128

26.3.3 MaxSpeed 129
26.3.4 MaxAcceleration 129
26.3.5 MaxJerk 130

27 MCAxis.Mechanism.LinearRotationRatio 131

27.1 Syntax for MCAxis.Mechanism.LinearRotationRatio 131
27.2 Properties of MCAxis.Mechanism.LinearRotationRatio 131
27.2.1 LinearRotationRatio 131

28 MCAxis.Mechanism.Monitoring 133

28.1 Syntax for MCAxis.Mechanism.Monitoring 133
28.2 Properties of MCAxis.Mechanism.Monitoring 133
28.2.1 IsReferenceSwitchEnabled 133
28.2.2 IsHWLimitSwitchMinActive 134
28.2.3 IsHWLimitSwitchMaxActive 134

29 MCAxis.Mechanism.Slope 135

29.1 Syntax for MCAxis.Mechanism.Slope 135
29.2 Properties of MCAxis.Mechanism.Slope 135
29.2.1 SlopeState 136
29.2.2 IsAccerlerating 136
29.2.3 IsAtConstantSpeed 136
29.2.4 IsDecelerating 137
29.2.5 IsStopped 137
29.2.6 SlopeType 137
29.2.7 ActualMaxSpeed 138
29.2.8 ActualMaxAcceleration 138
29.2.9 ActualMaxDeceleration 139
29.2.10 AccelerationSetpoint 139
29.2.11 SpeedSetpoint 139
29.2.12 Speed 140
29.2.13 Acceleration 140
29.2.14 Deceleration 141
29.2.15 Config.RampBreakMode 141

30 MCAxis.Mechanism.Units 142

30.1 Syntax for MCAxis.Mechanism.Units 142
30.2 Functions of MCAxis.Mechanism.Units 142
30.2.1 SetLinearUnits() 142
30.2.2 SetRotaryUnits() 143
30.3 Properties of MCAxis.Mechanism.Units 144
30.3.1 LinearPositionUnit 144

30.3.2 LinearSpeedUnit 145
30.3.3 LinearAccelerationUnit 145
30.3.4 LinearJerkUnit 146
30.3.5 RotaryPositionUnit 146
30.3.6 RotatorySpeedUnit 147
30.3.7 RotatoryAccelerationUnit 147
30.3.8 RotatoryJerkUnit 147
31 MCAxis.MoveHalt 149
31.1 Syntax for MCAxis.MoveHalt 149
31.2 Functions of MCAxis.MoveHalt 149
31.2.1 Start() 149
31.3 Example task MCAxis.MoveHalt 151
32 MCAxis.MoveHome 152
32.1 Syntax for MCAxis.MoveHome 152
32.2 Functions of MCAxis.MoveHome 152
32.2.1 Start() 152
32.2.2 SetReference() 154
32.2.3 SetAbsEncoderReference() 155
32.2.4 ClearAbsEncoderReferenceStatus() 156
32.3 Properties of MCAxis.MoveHome 156
32.3.1 Direction 157
32.3.2 SwitchConfig 158
32.3.3 Mode 159
32.3.4 Acceleration 160
32.3.5 SpeedSwitchSearch 160
32.3.6 MaxDistanceSwitchSearch 161
32.3.7 SpeedReferenceSearch 162
32.3.8 MaxDistanceReferenceSearch 162
32.3.9 ReferenceOffset 163
32.3.10 NormalPosition 164
32.3.11 ReferenceState 164
32.3.12 IsReferenceRunStarted, IsSwitchFound, IsReferenceFound, IsReferenceSet 165
32.3.13 IsAbsEncoderReferenceValid 166
32.3.14 ReferenceShift 166
33 MCAxis.MovePtp 168
33.1 Syntax for MCAxis.MovePtp 168
33.2 Functions of MCAxis.MovePtp 168
33.2.1 Start() 168

33.2.2 StartNewTargetPosition() 171
33.2.3 NewSpeed() 173
33.2.4 NewAcceleration() 174
33.2.5 NewDeceleration() 175
33.3 Properties of MCAxis.MovePtp 176
33.3.1 PositioningMode 176
33.3.2 TargetPosition 178
33.3.3 TargetWindow 178
33.4 Example task MCAxis.MovePtp 180
34 MCAxis.MoveVelocity 182
34.1 Syntax for MCAxis.MoveVelocity 182
34.2 Functions of MCAxis.MoveVelocity 182
34.2.1 Start() 182
34.2.2 Reverse() 185
34.2.3 NewSpeed() 185
34.2.4 NewAcceleration() 186
34.2.5 NewDeceleration() 187
34.3 Properties of MCAxis.MoveVelocity 188
34.3.1 Direction 188
34.4 Example task MCAxis.MoveVelocity 189
35 MCAxis.Position 190
35.1 Syntax for MCAxis.Position 190
35.2 Functions of MCAxis.Position 190
35.2.1 AbsEncOverflowHandler.Activate() 190
35.2.2 AbsEncOverflowHandler.Deactivate() 191
35.3 Properties of MCAxis.Position 191
35.3.1 Target 192
35.3.2 Actual 192
35.3.3 Setpoint 192
35.3.4 EncoderPosition 193
35.3.5 AcutalTrackingError 193
35.3.6 TrailingIndicatorMax 193
35.3.7 TrailingIndicatorMin 194
35.3.8 AbsEncOverflowHandler.IsActive 194
36 MCAxis.Power 195
36.1 Syntax for MCAxis.Power 195
36.2 Functions of MCAxis.Power 195
36.2.1 Enable() 195

36.2.2 Disable() 196
36.2.3 QuickStop() 197
36.3 Example task MCAxis.Power 198
37 MCAxis.Settings 199
37.1 Syntax for MCAxis.Settings 199
37.2 Functions of MCAxis.Settings 199
37.2.1 SaveToFile() 199
37.2.2 LoadFromFile() 200
38 MCAxis.State 201
38.1 Syntax for MCAxis.State 203
38.2 Properties of MCAxis.State 204
38.2.1 Value 204
38.2.2 IsDisabled 204
38.2.3 IsEnabled 205
38.2.4 IsRunning 205
38.2.5 IsSuspended 205
38.2.6 IsDisabledError 206
38.2.7 IsEnabledError 206
38.2.8 IsInactive 206
38.2.9 OperatingMode.Value 206
38.2.10 TypeOfMotion.Value 209
38.3 Example task MCAxis.State 212
39 MCAxis.State.Techno 213
39.1 Syntax for MCAxis.State.Techno 213
39.2 Properties of MCAxis.State.Techno 213
39.2.1 TechnoID 213
39.2.2 SyncState 214
39.2.3 IsDecoupled 214
39.2.4 IsDecoupling 214
39.2.5 IsCoupling 215
39.2.6 IsCoupled 215
39.2.7 SyncPosition 215
39.2.8 Cam.CamID 216
39.2.9 Cam.CamSegmentID 216
39.2.10 Cam.MasterOffset 217
39.2.11 Cam.FollowerOffset 217
39.2.12 Cam.MasterFactor 218
39.2.13 Cam.FollowerFactor 218

40 MCAxis.State.Transitions 220
40.1 Syntax for MCAxis.State.Transitions 222
40.2 Functions of MCAxis.State.Transitions 223
40.2.1 Enable() 223
40.2.2 Disable() 223
40.2.3 ErrorDisable() 223
40.2.4 ErrorStop() 224
40.2.5 ErrorLockUserInput() 225
40.2.6 Suspend() 225
40.2.7 Resume() 226
40.2.8 UserPositionReset() 227
40.3 Properties of MCAxis.State.Transitions 227
40.3.1 ErrorDisableMode 228
40.3.2 ErrorStopMode 228
41 MCAxis.Stop 229
41.1 Functions of MCAxis.Stop 229
41.1.1 Stop() 229
41.2 Example task MCAxis.Stop 230
42 MCAxis.TouchProbe 232
42.1 Syntax for MCAxis.TouchProbe 235
42.2 Functions of MCAxis.TouchProbe 235
42.2.1 Configure() 235
42.2.2 Enable() 236
42.2.3 Disable() 237
42.2.4 ReadConfiguration() 237
42.2.5 ReadState() 238
42.3 Example task MCAxis.TouchProbe 240
43 MCCouplingGroup 242
43.1 Syntax for MCCouplingGroup 242
43.2 Functions of MCCouplingGroup 242
43.2.1 AddFollowerAxis() 243
43.2.2 Redefine() 243
43.2.3 Undefine() 244
43.3 Properties of MCCouplingGroup 244
43.3.1 FollowerCount 244
43.3.2 MasterAxis 245
43.3.3 FollowerAxis[] 245
43.3.4 Name 245

44 MCGantryGroup 247
44.1 Syntax for MCGantryGroup 247
44.2 Functions of MCGantryGroup 248
44.2.1 AddFollowerAxis() 248
44.2.2 Redefine() 248
44.2.3 Undefine() 249
44.3 Properties of MCGantryGroup 249
44.3.1 GantryWindow[i] 250
44.3.2 ConfigGantryWindow[i] 250
44.3.3 FollowerCount 250
44.3.4 MasterAxis 251
44.3.5 FollowerAxis[i] 251
44.3.6 Name 251
44.4 Example task MCGantryGroup 252
45 MCGeo.Activate, MCGeo.Deactivate 254
45.1 Syntax for MCGeo.Activate, MCGeo.Deactivate 254
45.2 Functions of MCGeo.Activate, MCGeo.Deactivate 254
45.2.1 Activate() 254
45.2.2 Deactivate() 255
46 MCGeo.Buffer 257
46.1 Syntax for MCGeo.Buffer 257
46.2 Functions of MCGeo.Buffer 257
46.2.1 Clear() 257
46.2.2 Suspend() 258
46.2.3 Resume() 259
46.2.4 UserIndex.Enable() 259
46.2.5 UserIndex.Disable() 260
46.2.6 UserIndex.Reset() 260
46.3 Properties of MCGeo.Buffer 261
46.3.1 UserIndex.Actual 261
47 MCGeo.Coordinates 262
47.1 Syntax for MCGeo.Coordinates 262
47.2 Properties of MCGeo.Coordinates 262
47.2.1 Actual 262
47.2.2 Setpoint 263
47.2.3 Target 263
48 MCGeo.Coupling 265
48.1 Syntax for MCGeo.Coupling 266

48.2 Functions of MCGeo.Coupling 266
48.2.1 Couple() 266
48.2.2 Decouple() 268
48.2.3 GetPositioningMode() 269
48.2.4 GetCoupleType() 269
48.2.5 GetCoupleMode() 269
48.2.6 GetDecoupleMode() 270
48.3 Properties of MCGeo.Coupling 270
48.3.1 CoupleState 270
49 MCGeo.Diagnostics 271
49.1 Syntax for MCGeo.Diagnostics 271
49.2 Functions of MCGeo.Diagnostics 271
49.2.1 ClearErrors(), Software.ClearErrors() – path group 271
49.2.2 Software.GetErrorCode() 272
49.2.3 Software.GetMessageCode() 272
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 Functions 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 functionality 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 Cartesian2D 282
52.2.2 Cartesian3D 283
52.2.3 Scara 284
52.3 Configuration functions 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.Ellbow.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 IsAccerlerating 300
54.2.3 IsAtConstantSpeed 300
54.2.4 IsDecelerating 301
54.2.5 IsStopped 301
54.2.6 ActualMaxSpeed 301
54.2.7 ActualMaxAcceleration 302
54.2.8 SpeedSetpoint 302
54.2.9 AccelerationSetpoint 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 OrientationUnit 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 Functions 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 PositioningMode 315
57.3.2 BlockEnd 316
57.3.3 CoordinateSystemPlane 317
57.3.4 TravelAngle 318
57.3.5 TargetPositionHelixAxis 318
57.3.6 SmoothingDistance 318
57.3.7 Delay 319
57.3.8 TargetWindow 319
57.3.9 CenterPoint 319
57.3.10 TargetOrientation 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 PositioningMode 324
58.3.2 BlockEnd 325
58.3.3 SmoothingDistance 326

58.3.4 Delay 327
58.3.5 TargetWindow 327
58.3.6 IntermediatePoint 327
58.3.7 TargetPosition 328
58.3.8 TargetOrientation 328
59 MCGeo.MoveLinear 330
59.1 Syntax for MCGeo.MoveLinear 330
59.2 Functions of MCGeo.MoveLinear 330
59.2.1 Start() 330
59.3 Properties of MCGeo.MoveLinear 332
59.3.1 PositioningMode 332
59.3.2 BlockEnd 333
59.3.3 SmoothingDistance 334
59.3.4 Delay 335
59.3.5 TargetWindow 335
59.3.6 TargetCoordinates 335
59.3.7 TargetPosition 336
59.3.8 TargetOrientation 336
60 MCGeo.MoveOrientation 338
60.1 Syntax for MCGeo.MoveOrientation 338
60.2 Functions of MCGeo.MoveOrientation 338
60.2.1 Start() 338
60.3 Properties of MCGeo.MoveOrientation 340
60.3.1 PositioningMode 340
60.3.2 BlockEnd 341
60.3.3 Speed 341
60.3.4 Acceleration 342
60.3.5 Jerk 342
60.3.6 Delay 342
60.3.7 TargetWindow 343
60.3.8 TargetOrientation 343
61 MCGeo.MovePtp 344
61.1 Syntax for MCGeo.MovePtp 344
61.2 Functions of MCGeo.MovePtp 344
61.2.1 Start() 344
61.3 Properties of MCGeo.MovePtp 346
61.3.1 PositioningMode 346
61.3.2 BlockEnd 347

61.3.3 Delay 348
61.3.4 TargetWindow 348
61.3.5 TargetPosition 348
61.3.6 TargetOrientation 349
61.3.7 TargetCoordinates 349
62 MCGeo.MoveSpline 351
62.1 Syntax for MCGeo.MoveSpline 351
62.2 Functions of MCGeo.MoveCircle2D 351
62.2.1 Start() 351
62.3 Properties of MCGeo.MoveSpline 354
62.3.1 NodeCount 354
62.3.2 PositioningMode 354
62.3.3 BlockEnd 356
62.3.4 StartCondition 356
62.3.5 TargetCondition 357
62.3.6 SmoothingDistance 358
62.3.7 Delay 358
62.3.8 TargetWindow 359
62.3.9 VirtualStartPosition[x] 359
62.3.10 VirtualTargetPosition[x] 360
62.3.11 NodePosition[n] 360
62.3.12 NodeOrientation[n] 361
62.3.13 NodeCoordinates[n] 362
63 MCGeo.MoveSyncAxis 363
63.1 Syntax for MCGeo.MoveSyncAxis 363
63.2 Functions of MCGeo.MoveSyncAxis 363
63.2.1 Start() 363
63.2.2 GetTargetPosition() 364
63.2.3 SetTargetPosition() 365
63.2.4 GetPositioningMode() 365
63.2.5 SetPositioningMode() 366
64 MCGeo.Orientation 367
64.1 Syntax for MCGeo.Orientation 367
64.2 Properties of MCGeo.Orientation 367
64.2.1 Actual 367
64.2.2 Setpoint 368
64.2.3 Target 369
65 MCGeo.Position 370

65.1 Syntax for MCGeo.Position 370
65.2 Properties of MCGeo.Position 370
65.2.1 Actual 370
65.2.2 Setpoint 371
65.2.3 Target 371
66 MCGeo.Power 372
66.1 Syntax for MCGeo.Power 372
66.2 Functions of MCGeo.Power 372
66.2.1 Enable() 372
66.2.2 Disable() 373
66.2.3 QuickStop() 374
66.3 Example task MCGeo.Power 375
67 MCGeo.Settings 376
67.1 Syntax for MCGeo.Settings 376
67.2 Functions of MCGeo.Settings 376
67.2.1 SaveToFile() 376
67.2.2 LoadFromFile() 377
68 MCGeo.State 378
68.1 Properties of MCGeo.State 380
68.1.1 Value 380
68.1.2 IsDisabled 381
68.1.3 IsDisabledError 381
68.1.4 IsEnabled 382
68.1.5 IsEnabledError 382
68.1.6 IsInactive 382
68.1.7 IsRunning 383
68.1.8 IsSuspended 383
68.1.9 OperatingMode.Value 383
68.1.10 TypeOfMotion.Value 385
68.2 Example task MCGeo.State 388
69 MCGeo.State.Transitions 389
69.1 Syntax for MCGeo.State.Transitions 391
69.2 Functions of MCGeo.State.Transitions 391
69.2.1 Enable() 391
69.2.2 Disable() 392
69.2.3 Stop() 392
69.2.4 ErrorDisable() 393
69.2.5 ErrorStop() 394

69.2.6 ErrorLockUserInput() 394
69.2.7 Suspend() 395
69.2.8 ErrorSuspend() 396
69.2.9 Resume() 396
69.3 Properties of MCGeo.State.Transitions 397
69.3.1 ErrorDisableMode 397
69.3.2 ErrorStopMode 398
69.3.3 ErrorLevel 398
70 MCGeo.Stop 400
70.1 Syntax for MCGeo.Stop 400
70.2 Functions of MCGeo.Stop 400
70.2.1 Stop() 400
71 MCManager (MCMgr) 402
71.1 Syntax for MCMgr 402
71.2 Functions of MCMgr 402
71.2.1 CheckVersions() 402
71.2.2 GetDataSource() 403
71.2.3 WriteLogToFile() 403
71.2.4 GetAxisByID() 404
71.2.5 GetAxisByName() 404
71.2.6 GetTechnoByID() 404
71.2.7 GetTechnoByName() 405
71.2.8 GetGeoByID() 405
71.2.9 GetTechnoByName() 405
71.3 Properties of MCMgr 406
71.3.1 AxisCount 406
71.3.2 AxisList[] 406
71.3.3 TechnoCount 407
71.3.4 TechnoList[] 407
71.3.5 GeoCount 407
71.3.6 GeoList[] 408
71.3.7 FaultyObjectCount 408
71.3.8 FaultyObjectList[] 408
71.3.9 Name 410
71.3.10 Version 410
71.3.11 IsInitialized 410
72 MCTechno.Activate, MCTechno.Deactivate 411
72.1 Syntax for MCTechno.Activate, MCTechno.Deactivate 411

72.2 Functions of MCTechno.Activate, MCTechno.Deactivate 411
72.2.1 Activate() 411
72.2.2 Deactivate() 412
73 MCTechno.Coupling 414
73.1 Syntax for MCTechno.Coupling 414
73.2 Functions of MCTechno.Coupling 414
73.2.1 Couple() 414
73.2.2 Decouple() 416
73.2.3 GetCoupleMasterPosition() 417
73.2.4 GetCoupleFollowerPosition() 417
73.2.5 GetCoupleMasterDirection() 418
73.2.6 GetDecoupleDeceleration() 418
73.2.7 GetDecoupleMasterPosition() 418
73.2.8 GetDecoupleMode() 419
73.2.9 IsCoupling() 419
73.2.10 IsCoupled() 419
73.2.11 IsDecoupling() 420
73.2.12 IsDecoupled() 420
73.2.13 Filter.GetCoefficient(), Filter.SetCoefficient() 420
73.2.14 Gearing.Configure() 421
73.2.15 Gearing.GetFactor() 423
73.2.16 Gearing.GetDivisor() 423
73.2.17 Gearing.GetActivationMode() 423
73.3 Properties of MCTechno.Coupling 424
73.3.1 ExternalMaster.Offset 424
73.3.2 ExternalMaster.Smoothing 425
74 MCTechno.Coupling.Cam 427
74.1 Syntax for MCTechno.Coupling.Cam 427
74.2 Functions of MCTechno.Coupling.Cam 427
74.2.1 Create() 427
74.2.2 DefineSegment() 428
74.2.3 DefineSegmentAuto() 430
74.2.4 Activate() 431
74.2.5 ReadSegment() 432
74.2.6 Evaluate() 433
74.2.7 AnalyseSegment() 434
74.2.8 AnalyseSegmentDerivates() 436
75 MCTechno.Diagnostics 439

75.1 Syntax for MCTechno.Diagnostics 439
75.2 Functions of MCTechno.Diagnostics 439
75.2.1 ClearErrors(), Software.ClearErrors() – technology group 439
75.2.2 Software.GetErrorCode() 440
75.2.3 Software.GetMessageCode() 440
75.3 Properties of MCTechno.Diagnostics 441
75.3.1 Software.ErrorCount 441
75.3.2 IsErrorPending, Software.IsErrorPending 441
75.3.3 Software.MessageCount 442
75.3.4 IsMessagePending, Software.IsMessagePending 442
75.4 Example task for MCTechno.Diagnostics 442
76 MCTechno.Info 444
76.1 Syntax for MCTechno.Info 444
76.2 Properties of the functionality 444
76.2.1 Name 444
76.2.2 ObjectID 444
76.2.3 ClassID 445
76.3 Example task MCTechno.Info 445
77 MCTechno.Mechanism.Slope 446
77.1 Syntax for MCTechno.Mechanism.Slope 446
77.2 Functions of MCTechno.Mechanism.Slope 446
77.2.1 GetSlopeType(), SetSlopeType() 446
77.2.2 GetSpeed(), SetSpeed() 447
77.2.3 GetAcceleration(), SetAcceleration() 448
77.2.4 GetDeceleration(), SetDeceleration() 448
77.3 Properties of MCTechno.Mechanism.Slope 449
77.3.1 SlopeState 449
77.3.2 IsAccerlerating 450
77.3.3 IsAtConstantSpeed 450
77.3.4 IsDecelerating 450
77.3.5 IsStopped 450
78 MCTechno.Members 452
78.1 Syntax for MCTechno.Members 452
78.2 Properties of MCTechno.Members 452
78.2.1 FollowerCount 452
78.2.2 Follower[n] 452
78.2.3 FollowerCamCount 453
78.2.4 FollowerCam[n] 453

78.2.5 FollowerGearCount 453
78.2.6 FollowerGear[n] 454
78.2.7 Master 454
79 MCTechno.MoveHalt 455
79.1 Syntax for MCTechno.MoveHalt 455
79.2 Functions of MCTechno.MoveHalt 455
79.2.1 Start() 455
80 MCTechno.MovePtp 458
80.1 Syntax for MCTechno.MovePtp 458
80.2 Functions of MCTechno.MovePtp 458
80.2.1 Start() 458
80.2.2 StartNewTargetPosition() 461
80.2.3 NewSpeed() 462
80.2.4 NewAcceleration() 463
80.2.5 NewDeceleration() 464
80.2.6 GetPositioningMode(), SetPositioningMode() 465
80.2.7 GetTargetPosition(),SetTargetPosition() 467
80.2.8 GetTargetWindow(), SetTargetWindow() 468
80.2.9 StartSuperPose() 469
80.2.10 StartNewTargetPositionSuperPose() 470
81 MCTechno.MoveVelocity 472
81.1 Syntax for MCTechno.MoveVelocity 472
81.2 Functions of MCTechno.MoveVelocity 472
81.2.1 Start() 472
81.2.2 Reverse() 475
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 Functions 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 Functions 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 Functions of MCTechno.Stop 509
86.2.1 Stop() 509
Glossary 511

1 Introduction

1.1 Information about this document

This document is part of the product and must be read and understood before using the device. It contains important and safety-relevant information for operating the product properly and as intended.

For further information refer to the following information products:

– Application Notes
  Technical reports and application examples
– Application-oriented manuals
  Product-independent documentation

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** functionality allows you to query various information about an MCX axis that results from its configuration in the JetSym programming tool.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

2.1 Syntax for MCAxis.AxisConfigData

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.AxisConfigData.<value access>;
```

2.2 Properties of MCAxis.AxisConfigData

The functionality provides 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 designates the axis object that JetSym creates for programming.

Syntax

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

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

2.2.2 AxisMode

Description

Data type	Meaning
Type MCAxisModes	Defines the axis type.
MCAxisModes.InternalAxis	Fully functional real axis with servo amplifier and motor.
MCAxisModes.ExternalAxis	Axis that only has an actual position. It can be used as a master axis in a technology group. It is defined via the second encoder of a real axis.
MCAxisModes.SimulatedAxis	Axis that behaves like an internal axis, but whose servo amplifier and motor are simulated. This configuration allows you to create and test a program while the real hardware components are not yet available.
MCAxisModes.VirtualAxis	Axis that is calculated in the MCX kernel. It serves as an ideal master axis for technology groups.
MCAxisModes.ShadowAxis	Shadow axis that is primarily used to create cascaded technology groups. It always adopts the setpoint and actual position of another (virtual or real) axis. A shadow axis must always be the master axis in a technology group.

Syntax

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

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

2.2.3 AxisDriveType

Description

This property allows you to read the basic characteristics of the MCX axis. This information is used in loop commands across all axes to determine whether a command makes sense for this 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

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

State management

The property is used for diagnostics. It can only be accessed for reading. 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 ID of the MCX axis with respect to all axis objects of the MCX axes created in the JetSym hardware manager. Each axis object is assigned a unique number.

Syntax

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

State management

The property is used for diagnostics. It can only be accessed for reading. 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 with respect to the bus system. This number is assigned by the JetSym hardware manager for internal communication. Each bus device is assigned a unique DeviceID. Axes that are not bus devices (e.g. virtual axes) return the value `-1`.

Syntax

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

State management

The property is used for diagnostics. It can only be accessed for reading. The state of the axis remains unchanged.

The property can be read in any state.

2.2.6 IsSimulation

Description

Data type	Meaning
Bool	Reads whether the MCX axis has been configured as simulated. In projects in which an MCX axis is not used in every machine, this property allows the program to detect whether the axis in question is used in the configuration for this machine. `True`: The MCX axis is operated in simulation.

Syntax

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

State management

The property is used for diagnostics. It can only be accessed for reading. The state of the axis remains unchanged.

The property can be read in any state.

3 MCAxis.Buffer

The **MCAxis.Buffer** functionality allows you to switch the axis back and forth between buffered and normal mode. In addition, the functions and properties of this functionality allow you to manage the buffer and query the buffer status.

ⓘ Info

Axis behavior in buffered mode

The configuration and parameterization of an axis are locked in buffered mode.

After switching on, the axis is in *normal* operating mode, in which motion commands are "superseding". This means that a new motion command immediately replaces the travel movement that was active until then.

In *buffered* operating mode, motion commands for the axis are written to an MCX-internal buffer and executed one after the other. The condition for the transition from one travel movement to the next (block change condition) depends on the type of motion command. The following block change conditions apply in detail:

Motion command	Condition
MCAxis.MovePtp()	Setpoint reaches the target position (interpolator stop).
MCAxis.MoveVelocity()	Setpoint reaches the target speed.
MCAxis.MoveHome()	Setpoint reaches the reference position (interpolator stop).
MCAxis.MoveHalt()	The setpoint speed reaches 0 (interpolator stop).

Tab. 1: Overview of block change conditions

MCX axes can process the buffer in two ways:

– *Normal*

  The buffer is run through once. The system then waits for new motion commands.

– *Cyclic*

– After each run, buffer processing restarts with the first motion command and only ends when it is explicitly aborted (Delete()).

3.1 Syntax for MCAxis.Buffer

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Buffer.<function>(optional <parameter>);
```

3.2 Functions of MCAxis.Buffer

The functionality provides the following functions:

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

3.2.1 Create()

The function activates the buffer for this axis. As a result, the operating mode of the axis changes from *normal* to *buffered*.

Parameters

Parameter	Data type	Meaning
MCBufferTypes	MCBufferTypes (optional)	Defines the buffer type.
	MCBufferTypes.Normal (Active after reset)	Defines a buffer whose content is run through once.
	MCBufferTypes.Cyclic	Defines a buffer whose content is run through repeatedly from the beginning until processing is explicitly aborted.

Syntax

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

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Before calling further commands for the axis, query in the program whether the new operating state has been reached, as follows:

```
When AX1.Buffer.BufferState == MCBufferStates.Normal Continue;
```

or

```
When AX1.Buffer.BufferState == MCBufferStates.Cyclic Continue;
```

3.2.2 Clear()

The function deactivates buffer processing with the following effects:

– The operating mode of the axis changes from *buffered* to *normal*.

– The buffer is cleared.

– A movement that has already started is completed.

The commands `Power.Disable()` and `Power.Quickstop()` also deactivate the buffer.

Parameters

This function has no parameters.

Syntax

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

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

3.2.3 Delete()

The function is only relevant for the cyclic buffer type. It tells the system where the cyclic buffer content ends. This means that when processing reaches this command, it restarts processing the buffer at the command that was written after `Buffer.Create()`.

Parameters

This function has no parameters.

Syntax

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

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

3.2.4 Suspend()

The function stops buffer processing at the next block transition with the following effects:

– The current movement is completed.
– The next movement written to the buffer is no longer started.
– The axis remains in *buffered* operating mode.

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

Parameters

This function has no parameters.

Syntax

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

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

3.2.5 Resume()

The function resumes buffer processing that was stopped with Suspend() [▶ 35].

Parameters

This function has no parameters.

Syntax

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

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

3.2.6 Wait()

The function sets a waiting time in the buffer that delays the execution of a command relative to the command written to the buffer immediately before it.

Parameters

Parameter	Data type	Meaning
WaitingTime	Integer (optional)	Defines the waiting time in [ms].

Syntax

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

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

3.3 Properties of MCAxis.Buffer

The functionality provides 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 switching on, the axis is in *normal* operating mode, i.e. in non-buffered mode. The buffer status is `0`.

Data type	Meaning
MCBufferStates	Buffer status of the MCX axis:
0	The axis is in *normal* operating mode: no buffered mode.
Normal (1)	The axis is in buffer status *normal*: The buffer content is run through once.
Cyclic (2)	The axis is in buffer status *cyclic*: The buffer content is run through continuously.

Syntax

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

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

3.3.2 IsCyclic

Description

The property queries 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 buffer status *cyclic*: The buffer content is run through continuously.
False	– The axis is not in buffered mode. – The axis is in buffer status *normal*.

Syntax

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

State management

The property can be queried in any state.

3.3.3 IsNormal

Description

The property queries 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 buffer status *normal*: The buffer content is run through once.
False	– The axis is not in buffered mode. – The axis is in buffer status *cyclic*.

Syntax

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

State management

The property can be queried in any state.

3.4 Example task for MCAxis.Buffer

Here you will find an example task for a single axis that is operated with buffer type *cyclic*.

A cyclic buffer is created that starts reversing an axis between position 1000 and 0 at the speed given by the variable `Speed`. At each target, the axis stops for 200 ms.

```
Task Sample3 Autorun

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

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

    // Fill the buffer with commands for reversing mode
    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 cyclic buffer to the
MCX

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

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

```
    // After another 20 seconds, end reversing mode and
    // disable the axis once the last movement has finished
    delay(t#20s);
AX1.Buffer.Clear();
    When (AX1.Buffer.BufferState == 0) AND (Not AX1.State.IsRunning)
Continue;

    AX1.Power.Disable();
    When AX1.State.IsDisabled Continue;

End_Task;
```

4 MCAxis.Diagnostics

The **MCAxis.Diagnostics** functionality allows you to manage the error handling of an MCX axis.

ⓘ Info

Scope of the functionality

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 with the **MCAxis.Drive.Diagnostics** functionality.

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

The meaning of the error and message numbers is listed in the JetSym online help under "MotionControl errors".

4.1 Syntax for MCAxis.Diagnostics

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Diagnostics.<function or value access>(optional <parameter>);
```

4.2 Functions of MCAxis.Diagnostics

The functionality provides the following functions:

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

4.2.1 ClearErrors()

For errors, the function acknowledges an error state. The entries in the error buffer are cleared.

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

Parameters

This function has no parameters.

Syntax

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

ⓘ Info

Scope of the command

This command clears all errors assigned to the axis. These can be errors reported by the servo amplifier (drive) as well as errors of the motion control part in the control unit (MCX).

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state	
Disabled	–	Disabled	
Enabled	–	Enabled	
Running	–	Running	
Suspended	–	Suspended	
Error: enabled	√	Enabled	The system only changes to this state when the cause of the error has been eliminated. Otherwise, the error state may persist or be reached again immediately.
Error: disabled	√	Disabled	

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

4.2.2 GetErrorCode()

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

Parameters

Parameter	Data type	Meaning
BufferIndex	Integer (optional)	`BufferIndex` passes the position in the error buffer whose error number is to be read. The maximum value of the `BufferIndex` parameter can be read via the `ErrorCount` property.

Syntax

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

State management

The function can be read in any state.

4.2.3 GetMessageCode()

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

Parameters

Parameter	Data type	Meaning
BufferIndex	Integer (optional)	`BufferIndex` passes the position in the message buffer whose message number is to be read. The maximum value of the `BufferIndex` parameter can be read via the `MessageCount` property.

Syntax

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

State management

The function can be read in any state.

4.3 Properties of MCAxis.Diagnostics

The functionality provides the following properties:

- ErrorCount
- IsErrorPending
- MessageCount
- IsMessagePending

4.3.1 ErrorCount

Description

Property	Data type	Meaning
ErrorCount	Integer	Reads the number of errors recorded since the `ClearErrors()` function was last executed or since the control unit was restarted.

Syntax

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

State management

The property can be read in any state.

4.3.2 IsErrorPending

Description

Data type	Meaning
Bool	Queries whether an error is pending in the MCX system. This is equivalent to comparing `ErrorCount` with 0.

Syntax

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

State management

The property can be queried in any state.

4.3.3 IsMessagePending

Description

Data type	Meaning
Bool	Queries whether a message is pending in the MCX system. This is equivalent to comparing `MessageCount` with `0`.

Syntax

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

State management

The property can be queried in any state.

4.3.4 MessageCount

Description

Data type	Meaning
Integer	Reads the number of messages recorded since the `ClearErrors()` function was last executed or since the control unit was restarted.

Syntax

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

State management

The property can be read in any state.

4.4 Example task for MCAxis.Diagnostics

Here you will find an example task for a single axis that contains various Motion API functions:

```
Var
    AxisError: 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 the error variables used for display
    For i := 0 To 9 Do
        AxisError[i] := 0;
    End_For;
    AcknowledgeError := False;

    When AX1.Diagnostics.IsErrorPending Continue;

        Taskbreak Automatic;
        // insert further error handling here

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

    // Errors have been eliminated
    When AcknowledgeError Continue;
```

```
        AX1.Diagnostics.ClearErrors();
        Taskrestart Automatic;
End_Loop;

End_Task;
```

5 MCAxis.Drive.Amplifier

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the "Amplifier" commissioning page. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API functionalities also allow you to change or read these parameters from the STX program.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

5.1 Syntax for MCAxis.Drive.Amplifier

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Amplifier.<value access>;
```

5.2 Properties of MCAxis.Drive.Amplifier

The functionality provides 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 management

The properties can only be accessed for reading, as 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 management

The properties can only be accessed for reading, as 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 current internal temperature of the servo amplifier.

Syntax

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

State management

The properties can only be accessed for reading, as the internal 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 regenerative energy during braking. If it exceeds the value defined here, an overvoltage error is triggered, which prevents any further rise.

Data type	Meaning
Double (optional)	Reads the error trigger threshold for overvoltage in the DC link of the servo amplifier in [V_{dc}].

Syntax

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

State management

The property can only be accessed for reading, as this error trigger 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 becomes too low due to insufficient mains voltage or excessive load, an error is triggered. This property allows you to read the error threshold. For some device types, the error threshold can also be adjusted, but this is only recommended in very few exceptional cases.

Data type	Meaning
Double (optional)	Reads or writes the error trigger threshold for undervoltage in the DC link of the servo amplifier in [V_{dc}].

Syntax

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

State management

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, as the servo amplifier is optimized for this frequency. The power loss of the servo amplifier changes with the setting of this property.

In a few cases, it can make sense to change this value:

– With very high loads and low speeds, a lower value can make sense.

– With very high speeds and/or extremely dynamic positioning, a higher value can make sense.

This property cannot be changed on all devices.

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

Syntax

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

State management

The property can be read and written in any state.

5.2.7 PwmFrequencyReduction

Description

On some devices, this property can be activated to automatically reduce the PWM frequency at very low motor speeds. In exceptional cases, this can make sense to reduce the power loss of the servo amplifier at high loads and very low speeds.

This property cannot be changed on all devices.

Data type	Meaning
DrivePwmFrequenciesReductionModes (optional)	Reads or writes the reduction of the PWM frequency of the inverter in the servo amplifier. Frequency options as specified.
DrivePwmFrequenciesReductionModes.Disable	Deactivates the property.
DrivePwmFrequenciesReductionModes.Enable	Activates the property.

Syntax

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

State management

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 management

The property can be read in any state.

5.2.9 RatedDCLinkVoltage

Description

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

Syntax

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

State management

The property can be read in any state.

5.2.10 WireTestMode

Description

Executing the property can take a short time. If it therefore cannot be used in the application, the function can be configured.

Not all devices support the motor cable test.

Data type	Meaning
DriveWireTestModes (optional)	Reads or writes whether the servo amplifier performs a motor cable test when the motor function is enabled for the first time.
DriveWireTestModes.Disable	Deactivates the property.
DriveWireTestModes.Enable	Activates the property.

Syntax

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

State management

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 integration value of the I²t monitoring of the servo amplifier assigned to this axis in [%]

Syntax

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

State management

The property can be read in any state.

6 MCAxis.Drive.Brake

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the "Brake" commissioning page. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API functionalities also allow you to change or read these parameters from the STX program.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

6.1 Syntax for MCAxis.Drive.Brake

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Brake.<function or value access>;
```

6.2 Functions of MCAxis.Drive.Brake

The functionality provides the following functions:

– Apply()
– Release()

6.2.1 Apply()

Description

The function applies the motor holding brake. It only takes effect if the `EncDriv.Drive.Brake.Mode` property (motor brake operating mode) is set to *manual*. In *auto* operating mode, the servo amplifier controls the brake according to the drive state.

Parameters

This function has no parameters.

Syntax

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

State management

The function can be called in any state.

6.2.2 Release()

Description

The function releases the motor holding brake. It only takes effect if the `EncDriv.Drive.Brake.Mode` property (motor brake operating mode) is set to *manual*. In *auto* operating mode, the servo amplifier controls the brake according to the drive state.

Parameters

This function has no parameters.

Syntax

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

State management

The function can be called in any state.

6.3 Properties of MCAxis.Drive.Brake

The functionality provides the following properties:

– ApplyDelay
– ReleaseDelay
– Mode
– State

6.3.1 ApplyDelay

Description

When the axis is disabled (*disable*), this delay causes the brake output to be switched first, 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 it is disabled, because the brake cannot take effect immediately due to the mechanical movement of the brake pads.

Data type	Meaning
Double	Reads or writes the delay time for applying the motor holding brake in [ms]. The delay only takes effect if the `EncDriv.Drive.Brake.Mode` property (motor brake operating mode) is set to *auto*.

Syntax

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

State management

The property can be read and written in any state.

6.3.2 ReleaseDelay

Description

When the axis is enabled (*enable*), this delay causes the brake output to be switched first, and the `isEnabled` message becomes active only after the delay time set here has elapsed. The output for the motor holding brake switches as soon as current is applied to the motor. Since the brake may not release immediately, the delay time ensures that the motor is not moved while the brake may not yet be fully released.

Data type	Meaning
Double	Reads or writes the delay time for releasing the motor holding brake in [ms]. The delay only takes effect if the `EncDriv.Drive.Brake.Mode` property (motor brake operating mode) is set to *auto*.

Syntax

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

State management

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 mode for the operation of the motor holding brake.

Data type	Meaning
DriveBrakeModes	Operating mode of the motor holding brake:
DriveBrakeModes.Manual	The motor holding brake is not managed by the servo amplifier. The user controls the brake directly via the program or the motion setup.
DriveBrakeModes.Auto	Select this operating mode in the following cases: – No brake is connected. – The brake is to be operated for service purposes while the motor is de-energized.

Syntax

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

State management

The property can be read and written in any state.

6.3.4 State

Description

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

Data type	Meaning
DriveBrakeStates	Position of the motor holding brake:
DriveBrakeStates.Applied	The motor holding brake is applied.
DriveBrakeStates.Released	The motor holding brake is released.

Syntax

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

State management

The property can be read in any state.

7 MCAxis.Drive.Control.Current

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the commissioning pages Control > Current/Torque and Limit values. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and, in some cases, adjust these parameters from the STX program.

In a servo amplifier, motor control is implemented in such a way that the motor current is proportional to the motor torque. For this reason, such a controller can be called both a current controller and a torque controller, although only the current is actually controlled. The torque values are calculated from the current values by multiplying them by the torque constant.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

7.1 Syntax for MCAxis.Drive.Control.Current

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Control.Current.<value access>;
```

7.2 Properties of MCAxis.Drive.Control.Current

The functionality provides the following properties:

- Kp
- Tn
- ActualMotorCurrent
- ActualMotorVoltage
- MaxMotorCurrent

7.2.1 Kp

Description

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

The current controller is designed as a PI controller. The setting of the current controller depends largely on the motor data and can therefore easily be determined by calculation. This is the preferred procedure during commissioning. The current controller can be set automatically in the motion setup of JetSym using the Controller optimization button.

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

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing. In most cases, however, this only makes sense during commissioning of the axis for tuning the control. **NOTICE! A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.**

7.2.2 Tn

Description

This property allows you to read or adjust the integral action time for the current controller of the servo amplifier.

The current controller is designed as a PI controller. The setting of the current controller depends largely on the motor data and can therefore easily be determined by calculation. This is the preferred procedure during commissioning. The current controller can be set automatically in the motion setup of JetSym using the Controller optimization button.

Data type	Meaning
Double	Defines the integral action time of the current controller in [ms].

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing. In most cases, however, this only makes sense during commissioning of the axis for tuning the control. **NOTICE! A change during machine operation can lead to uncontrolled movements if the written 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

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

State management

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 output by the servo amplifier in [V_{eff}].

Syntax

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

State management

The property can be read in any state.

7.2.5 MaxMotorCurrent

Description

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

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

- ► Avoid changes during machine operation.
- ► As a rule, changes only make sense during commissioning of the axis for tuning the control.

8 MCAxis.Drive.Control.Position

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the commissioning pages Control > Position and Feedforward. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and, in some cases, adjust these parameters from the STX program.

The position controller in a servo amplifier enables exact control of the motor position according to the set-point generation in the control unit. It is designed as a P controller with speed feedforward.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

8.1 Syntax for MCAxis.Drive.Control.Position

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Control.Position.<function or value access>;
```

8.2 Functions of MCAxis.Drive.Control.Position

The functionality provides the following functions:

– ResetTrailingIndicators()

8.2.1 ResetTrailingIndicators()

Description

Following error peak detection records the highest position following error that has occurred in the positive and negative direction. The `ResetTrailingIndicators()` function resets following error peak detection to zero and starts a new measurement.

Parameters

This function has no parameters.

Syntax

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

State management

The function can be called in any state.

8.3 Properties of MCAxis.Drive.Control.Position

The functionality provides 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 starts to oscillate.

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

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

- ► Avoid changes during machine operation.
- ► As a rule, changes only make sense during commissioning of the axis for tuning the control.

8.3.2 FeedbackSource

Description

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

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

Data type	Meaning
DriveFeedbackSources	Reads which position encoder is selected for position control or changes the selection:
DriveFeedbackSources.Encoder1	The position encoder at the first encoder interface is used.
DriveFeedbackSources.Encoder2	The position encoder at the second encoder interface is used.

Syntax

```
PositionEncoder1 := AX1.Drive.Control.Position.FeedbackSource;
AX1.Drive.Control.Position.FeedbackSource := DriveFeedbackSources.Encoder1;
```

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements when changing the position encoder during operation

This setting should be defined via the motion setup during initial commissioning of the axis. If it is necessary to switch between the two position encoders during machine operation, position jumps can occur, which in turn can lead to uncontrolled movements of the drive. In addition, the reference position can change when the position encoder is changed.

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

8.3.3 SpeedFeedForwardScale

Description

This property allows you to adjust the effect of the speed feedforward on the control of the MCX axis. The speed feedforward compensates for the steady-state deviation of the position controller (P controller).

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

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

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

► Avoid changes during machine operation.

► As a rule, changes only make sense during commissioning of the axis for tuning the control.

9 MCAxis.Drive.Control.Speed

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the commissioning pages Control > Speed, Limit values and Feedforward. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and, in some cases, adjust these parameters from the STX program.

The speed controller in a servo amplifier enables dynamic adjustment of the motor speed to the specifications of the position controller. It is designed as a PI controller with torque feedforward.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

9.1 Syntax for MCAxis.Drive.Control.Speed

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Control.Speed.<value access>;
```

9.2 Properties of MCAxis.Drive.Control.Speed

The functionality provides the following properties:

- Kp
- Tn
- Tf
- ActualValue
- Setpoint
- Isum
- IsumPreset
- OverloadFactor
- MaxMotorSpeed
- SetpointLimit
- DifferenceWindow
- TotalInertia
- TorqueFeedForwardScale

9.2.1 Kp

Description

This property allows you to read or adjust the gain 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 starts to oscillate.

Data type	Meaning
Double	Defines the gain factor of the speed controller in [Nm/1000 rpm].

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

- ► Avoid changes during machine operation.
- ► As a rule, changes only make sense during commissioning of the axis for tuning the control.

9.2.2 Tn

Description

This property allows you to read or adjust the integral action time for the speed controller of the servo amplifier.

The speed controller is designed as a PI controller. The lower the integral action time, the more dynamic the drive becomes. If the values are too low, the drive starts to oscillate. With high mechanical flywheel masses on the motor shaft, the integral action time generally has to be set higher.

Data type	Meaning
Double	Defines the integral action time of the speed controller in [ms].

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

- ► Avoid changes during machine operation.
- ► As a rule, changes only make sense during commissioning of the axis for tuning the control.

9.2.3 Tf

Description

This property allows you to read or adjust the filter time constant for the speed measurement of the servo amplifier.

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

The following dependencies apply:

– The higher the filter time constant, the smoother the actual value of the speed becomes.
– The higher the resolution of the position encoder, the lower the filter time constant can be set.
– Since the speed filter represents a dead time in 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 the speed measurement in [ms].

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

► Avoid changes during machine operation.
► As a rule, changes only make sense during commissioning of the axis for tuning the control.

9.2.4 ActualValue

Description

Data type	Meaning
Double	Reads the current motor speed determined via the first position encoder of the servo amplifier in [rpm].

Syntax

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

State management

The property can be read in any state.

9.2.5 Setpoint

Description

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

Syntax

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

State management

The property can be read in any state.

9.2.6 Isum

Description

Data type	Meaning
Double	Reads the calculated integral component of the speed controller or writes an application-specific value in [Nm]. This value is recalculated by the speed controller in every control cycle by adding the current control deviation.

Syntax

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

State management

The property can be read in any state.

NOTICE



Write access: Risk of uncontrolled movements

A change during machine operation can lead to uncontrolled movements if the written value does not match the load on the drive.

- ► Avoid changes during machine operation.
- ► As a rule, changes only make sense during commissioning of the axis for tuning the control.

A conceivable use case is the following: If an axis is deliberately moved into a blocking situation (e.g. for pressing), it can be helpful to clear the integral component once before positioning out of the blocked position is started.

9.2.7 IsumPreset

Description

This property allows you to read or adjust the preset value of the integral component of the PI speed controller at pulse enable of the servo amplifier.

This value acts like a "starting torque" at pulse enable. A non-zero value can make sense for vertical axes to prevent "sagging" when switching on. A suitable value for `IsumPreset` can be determined, for example, by reading the value of `Isum` while a vertical axis is at standstill before it is disabled.

Data type	Meaning
Double	Reads or writes the preset value for the torque at pulse enable [Nm].

Syntax

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

State management

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 speed setting of the servo amplifier.

Based on the rated speed of the motor, this property allows you to define the maximum permissible speed. Depending on the type of application, it can be above or below the rated value.

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

Syntax

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

State management

The property can be read and written in any state.

However, a changed value may only take effect after a (renewed) change to the **Enabled** state.

9.2.9 MaxMotorSpeed

Description

Data type	Meaning
Double	Reads the maximum available motor speed in [rpm]. The value is calculated from the rated speed of the motor and the overload factor.

Syntax

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

State management

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 refers to the maximum available motor speed (MaxMotorSpeed [▶ 63]).

Data type	Meaning
Double	Reads or writes the limitation of the setpoint speed in [%].

Syntax

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

State management

The property can be read and written in any state.

9.2.11 DifferenceWindow

Description

This property allows you to read or adjust the limit for monitoring the speed deviation. If the drive cannot follow the specification of the position controller and the deviation between setpoint and actual speed exceeds the value defined in this property, an error is triggered. In this way, unwanted blocking of a drive is monitored, for example.

Data type	Meaning
Double	Reads or writes the error threshold for the deviation of the actual speed from the setpoint speed in [%].

Syntax

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

State management

The property can be read and written in any state.

9.2.12 TotalInertia

Description

This property allows you to read or adjust the setting for the moment of inertia of the mechanics relative to the motor shaft. The value of this property is included as a factor in the calculation of the torque feedforward. Especially in very dynamic applications, the torque feedforward has an important influence on the control result of an MCX axis.

The torque feedforward determines a setpoint torque for the torque controller from the calculated setpoint acceleration multiplied by the scaling (TorqueFeedForwardScale [▶ 65]) and the moment of inertia (`TotalInertia`). This value is added to the output of the speed controller.

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

Syntax

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

State management

The property can be read and written in any state.

9.2.13 TorqueFeedForwardScale

Description

This property allows you to read or adjust the scaling of the torque feedforward effect. The value of this property is included as a factor in the calculation of the torque feedforward. Especially in very dynamic applications, the torque feedforward has an important influence on the control result of an MCX axis.

The torque feedforward determines a setpoint torque for the torque controller from the calculated setpoint acceleration multiplied by the scaling (`TorqueFeedForwardScale`) and the moment of inertia (TotalInertia [▶ 64]). This value is added to the output of the speed controller.

Data type	Meaning
Double	Reads or writes the scaling of the torque feedforward effect in [%].

Syntax

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

State management

The property can be read and written in any state.

10 MCAxis.Drive.Control.Torque

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the commissioning pages Control > Current/Torque, and Limit values. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and, in some cases, adjust these parameters from the STX program.

In a servo amplifier, motor control is implemented in such a way that the motor current is proportional to the motor torque. For this reason, such a controller can be called both a current controller and a torque controller, although only the current is actually controlled. The torque values are calculated from the current values by multiplying them by the torque constant.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

10.1 Syntax for MCAxis.Drive.Control.Current

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Control.Torque.<value access>;
```

10.2 Properties of MCAxis.Drive.Control.Torque

The functionality provides the following properties:

- ActualValue
- Setpoint
- OverloadFactor
- MaxMotorTorque
- SetpointLimit

10.2.1 ActualValue

Description

Data type	Meaning
Double	Reads the current motor torque determined from the measured motor current of the servo amplifier in [Nm]

Syntax

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

State management

The property can be read in any state.

10.2.2 Setpoint

Description

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

Syntax

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

State management

The property can be read in any state.

10.2.3 OverloadFactor

Description

Based on the rated torque of the motor, this property allows you to define the maximum permissible torque. Depending on the type of application, it can be above or below the rated value.

Data type	Meaning
Double	Reads or writes the overload factor for the torque setting of the servo amplifier in [%].

Syntax

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

State management

The property can be read and written in any state.

However, a changed value may only take effect after a (renewed) change to the **Enabled** state.

10.2.4 MaxMotorTorque

Description

Data type	Meaning
Double	Reads the maximum available motor torque in [Nm]. The value is calculated from the rated torque of the motor and the overload factor.

Syntax

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

State management

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. The value refers to the maximum available motor torque (MaxMotorTorque [▶ 67]).

Data type	Meaning
Double	Reads or writes the limitation of the setpoint speed in [%].

Syntax

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

State management

The property can be read and written in any state.

11 MCAxis.Drive.Diagnostics

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

Scope of the functionality

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, warnings do not. Errors must be acknowledged by the user in order to continue operating the drive, while warnings are cleared automatically as soon as the cause no longer exists.

The meaning of the error numbers is listed in the JetSym online help or in the operating instructions of the servo amplifier.

The `MCAxis.Diagnostics.ClearErrors()` command acknowledges all errors, including those of the servo amplifiers.

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

11.1 Syntax for MCAxis.Drive.Diagnostics

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Diagnostics.<function or value access>;
```

11.2 Functions of MCAxis.Drive.Diagnostics

The functionality provides the following functions:

– IsWarningPending()

11.2.1 IsWarningPending()

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

Parameters

Parameter	Data type	Meaning
WarningNr	DriveWarningTypes (optional)	
	DriveWarningTypes.AnyWarning (default)	The servo amplifier displays a warning.
	DriveWarningTypes.UndervoltageDCLink	– The DC link voltage is too low. – A DC link voltage is present while the control is disabled.
	DriveWarningTypes.OvertemperatureDrive	The temperature of the heat sink or of the device interior is close to the overtemperature shutdown threshold.
	DriveWarningTypes.OvertemperatureMotor	The motor temperature is close to the overtemperature shutdown threshold.
	DriveWarningTypes.STOactive	The STO function is active while the control is disabled.
	DriveWarningTypes.PowerSupply	A warning in the power supply is pending.
	DriveWarningTypes.I2TDrive	The I²t value for the device current is close to the shutdown threshold.
	DriveWarningTypes.I2TMotor	The I²t value for the motor current is close to the shutdown threshold.

Syntax

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

State management

The function can be called in any state.

11.3 Properties of MCAxis.Drive.Diagnostics

The functionality provides the following properties:

– AcutalError
– IsErrorPending

11.3.1 ActualError

Description

Property	Data type	Meaning
ActualError	dword	Reads the error number of an error pending at the servo amplifier. If the return value is `0`, the servo amplifier is free of errors.

Syntax

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

State management

The property can be read in any state.

11.3.2 IsErrorPending

Description

Data type	Meaning
Bool	Queries whether an error is pending at the servo amplifier of the MCX axis. `0`: The servo amplifier is free of errors. `1`: The servo amplifier displays an error. This is equivalent to comparing `ErrorCount` with `0`.

Syntax

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

State management

The property can be queried in any state.

12 MCAxis.Drive.DigitalInput

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the I/O commissioning page. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and, in some cases, adjust these parameters from the STX program.

In the MCX, the signals at the digital inputs are used for the functions of reference switches, limit switches and STO (safe torque off). You can query the state of the limit and reference switches via the properties of the MCAxis.Mechanism.Monitoring [▶ 133] functionality.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

12.1 Syntax for MCAxis.Drive.DigitalInput

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.DigitalInput.<value access>;
```

12.2 Properties of MCAxis.Drive.DigitalInput

The functionality provides 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 of the servo amplifier. This defines at which voltage at the digital input the reference switch is evaluated as active.

Data type	Meaning
DrivePolarityModes	Reads or writes the type of polarity setting.
DrivePolarityModes.HighActive	The reference switch is active when a 24 V signal is present at 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 management

The property can be read and written in any state.

12.2.2 LimitSwitchMode

Description

This property allows you to read or adjust the function of the hardware limit switches connected to the corresponding digital input of the servo amplifier. This defines whether an input signal is to 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 (evaluated).
DriveLimitSwitchModes.Max	Only the hardware limit switch in the positive travel direction is used (evaluated).

Syntax

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

State management

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 of the servo amplifier. This defines at which voltage at the digital input the limit switch is evaluated as active.

Data type	Meaning
DrivePolarityModes	Reads or writes the type of polarity setting.
DrivePolarityModes.HighActive	The hardware limit switch in the negative travel direction is active when a 24 V signal is present at 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

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

State management

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 of the servo amplifier. This defines at which voltage at the digital input the limit switch is evaluated as active.

Data type	Meaning
DrivePolarityModes	Reads or writes the type of polarity setting.
DrivePolarityModes.HighActive	The hardware limit switch in the positive travel direction is active when a 24 V signal is present at the digital input.
DrivePolarityModes.LowActive	The hardware limit switch in the positive travel direction is active when a 0 V signal is present at the digital input.

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing.

12.2.5 IsPulseEnabled

Description

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

The query returns `TRUE` if the corresponding digital inputs are wired so that the amplifier output stage is enabled, i.e. voltage pulses can be output to the motor.

Data type	Meaning
Bool	Reads the status of the STO function of the servo amplifier: `True`: The motor can be operated. `False`: The motor is safely disabled so that it cannot generate torque.

Syntax

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

State management

The property can be read in any state.

13 MCAxis.Drive.EncoderChannel

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the Encoder commissioning page. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and, in some cases, change these parameters from the STX program.

The servo amplifiers usually have 2 interfaces for a position encoder. This is why this functionality is available twice.

Position feedback is required for commutation as well as for speed and position control. Various types of position encoders can be connected. Which types are supported in your application can be found in the operating instructions of the servo amplifier.

The second position encoder can be used for 2 different functions:

1. As an additional position encoder for the axis, to compensate for any slip between material and motor.

   In some applications, it makes sense to mount a second position encoder directly on the load in addition to the motor position encoder. In this case, the motor position encoder is used for commutation and speed control, and the additional position encoder for position control. As a result, position control takes place directly at the load and not on the rotor position of the motor. Slip between motor shaft and load is thus compensated automatically.

   Procedure:

   For the parameter `AX1.Drive.Control.Position.FeedbackSource`, select the value `DriveFeedbackSources.Encoder2`, or in the motion setup of JetSym, on the page Control > Position, set the parameter Encoder for position control to Encoder 2.

2. As the position of an external axis in a technology group.

   Procedure:

   In the motion setup of JetSym, on the Axis parameters page, select the option Use encoder 2 as external axis.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

13.1 Syntax for MCAxis.Drive.EncoderChannel

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.EncoderChannel[1..2].<value access>;
```

13.2 Properties of MCAxis.Drive.EncoderChannel

The functionality provides 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 encoder connected to the corresponding digital input of the servo amplifier.

Data type	Meaning
DriveEncoderTypes	Reads or writes the type of the connected position encoder.
DriveEncoderTypes.Disabled	No position encoder is connected.
DriveEncoderTypes.Resolver	A resolver is connected.
DriveEncoderTypes.Hiperface	A position encoder with Hiperface interface is connected.
DriveEncoderTypes.EnDat	A position encoder with EnDat 2.2 interface is connected.
DriveEncoderTypes.RectangleIncremental-Encoder	An incremental encoder with square-wave signals is connected.
DriveEncoderTypes.SinCosIncrementalEncoder	An incremental encoder with sine signals is connected.
DriveEncoderTypes.HiperfaceDSL	A position encoder with Hiperface DSL interface is connected.
DriveEncoderTypes.LinMot	A LinMot drive is connected.

Syntax

```
PositionEncoder1 := AX1.Drive.EncoderChannel[1].EncType;
AX1.Drive.EncoderChannel[1].EncType := DriveEncoderTypes.Resolver;
```

State management

The property can be read and written in any state.

However, a changed value only takes effect at the state transition from **Disabled** to **Enabled**.

13.2.2 EncLines

Description

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

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

Syntax

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

State management

The property can be read and written in any state.

However, a changed value only takes effect at the state transition from **Disabled** to **Enabled**.

13.2.3 GearMotor

Description

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

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

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

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing.

13.2.4 GearEncoder

Description

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

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

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

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing.

13.2.5 MultiTurnPosition

Description

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

Data type	Meaning
Integer	Reads the value of the revolution counter of the position encoder in [rev]. The value is only non-zero for absolute encoders.

Syntax

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

State management

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 encoder interface of the servo amplifier.

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

Syntax

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

State management

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 position encoder connected to the corresponding encoder interface of the servo amplifier.

The MCX axis reads the value from the absolute encoder; it 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 only non-zero for absolute encoders.

Syntax

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

State management

The property can be read in any state.

14 MCAxis.Drive.GenericAccess

The **MCAxis.Drive.GenericAccess** functionality allows you to access special parameters of the servo amplifier of an MCX axis.

For operating an axis via the MCX kernel from JetSym, all required parameters of the servo amplifier are normally standardized and made available in SI units via the motion setup and the Motion API functions.

In special cases, it can be helpful to access a parameter on a servo amplifier that cannot be reached via the Motion API commands. The Technical Support of Bucher Automation usually provides you with sample command sets for this purpose.

NOTICE



Risk of unwanted behavior of the servo amplifier!

Generic access to a parameter via the **MCAxis.Drive.GenericAccess** command only performs the desired function for the servo amplifiers of one product series! The command cannot automatically be transferred to other product series.

► Do not use the command with servo amplifiers of different product series.

In a Bucher-Automation control unit, generic access to a servo amplifier is performed via the CANopen protocol. This applies to communication buses based on CAN as well as on EtherCAT.

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

For the command to work correctly, the correct data type must also be used for the parameter addressed via *object* and *subindex*. For this reason, the **MCAxis.Drive.GenericAccess** functionality provides a separate function for reading and for writing data for each data type.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

14.1 Syntax for MCAxis.Drive.GenericAccess

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.GenericAccess.CANopen.<function or value access>(optional
<parameter 1, parameter 2, parameter n>);
```

14.2 Functions of MCAxis.Drive.GenericAccess

The functionality provides 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 with the data type `UInt8` from the servo amplifier.

Parameters

Parameter	Data type	Meaning
ObjectIndex	Int	Object address of the parameter in the servo amplifier.
SubIndex	Word	Subindex number of the CANopen object.

Same procedure for 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 the example of .ReadUInt8()

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

State management

The function can be called in any state.

14.2.2 CANopen.WriteUInt8()

Description

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

Parameters

Parameter	Data type	Meaning
ObjectIndex	Int	Object address of the parameter in the servo amplifier.
SubIndex	Word	Subindex number of the CANopen object.
Value	Byte	Value to be written to the object.

Same procedure for all data types

You write data to objects with a different data type in the same way. The data type of the `Value` parameter must match the command.

The available commands are:

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

Syntax using the example of .WriteUInt8()

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

State management

The function can be called in any state.

14.3 Properties of MCAxis.Drive.GenericAccess

The functionality provides the following properties:

– CANopen.OSVersion
– CANopen.IsAvailable

14.3.1 CANopen.OSVersion

Description

Data type	Meaning
String	Reads the firmware version of the servo amplifier of the MCX axis. For EtherCAT/ CANopen objects, the firmware version is read as a string.

Syntax

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

State management

The property can be read in any state.

14.3.2 CANopen.IsAvailable

Description

Data type	Meaning
String	Reads whether the MCX axis provides a CANopen interface.

Syntax

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

State management

The property can be read in any state.

15 MCAxis.Drive.InvertDirection

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

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

15.1 Syntax for MCAxis.Drive.InvertDirection

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

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

15.2 Properties of MCAxis.Drive.InvertDirection

The functionality provides 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.

With standard wiring, a positioning command in the positive direction usually causes the motor shaft to rotate clockwise (as seen by the observer). If the `MCAxis.Drive.InvertDirection` property is activated, a positioning command in the positive direction causes the motor shaft to rotate counterclockwise instead.

Data type	Meaning
DriveInvertDirectionModes	Reads or defines whether the inversion of the direction of rotation is active or inactive.
DriveInvertDirectionModes.Disabled	Direction inversion is deactivated.
DriveInvertDirectionModes.Enabled	Direction inversion is activated.

Syntax

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

State management

The property can be read in any state.

Although the property can also be written in any state, this **does not make sense** in the **enabled** and **running** states. Depending on the type of servo amplifier, write access in these states can result in the input being ignored until the next state change to **enable**, or in the servo amplifier outputting an error.

Since this is a property of the machine, it is not changed during operation. Instead, this property is stored in the control unit together with the other axis data and configured again for the axis when the control unit boots.

16 MCAxis.Drive.IsReadyForOperation

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

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

16.1 Syntax for MCAxis.Drive.IsReadyForOperation

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

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

16.2 Properties of MCAxis.Drive.IsReadyForOperation

The functionality provides the following properties:

– IsReadyForOperation

16.2.1 IsReadyForOperation

Description

It is recommended to query 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 free of errors.
– The DC link voltage is sufficiently high (mains voltage is switched on).
– The STO signal (safe torque off) is not active.
– A position encoder is configured.

Data type	Meaning
Bool	Reads whether the servo amplifier is ready to be switched on (**enabled** state).

Syntax

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

State management

The property can be read in any state.

17 MCAxis.Drive.Motor

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

These are essentially the parameters that are available for Bucher-Automation axes in the motion setup of the JetSym programming tool on the Motor commissioning page. When the control unit is initialized, the parameters are assigned the values saved during commissioning.

The Motion API commands also allow you to read and adjust these parameters from the STX program.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

17.1 Syntax for MCAxis.Drive.Motor

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.Motor.<function or value access>(optional <parameter>);
```

17.2 Functions of MCAxis.Drive.Motor

The functionality provides the following functions:

– MeasureCommutationOffset()

17.2.1 MeasureCommutationOffset()

The function performs a measurement of the commutation offset for the MCX axis. The parameter passed defines the measuring method.

Measuring the commutation offset is necessary if, after switching on, the system does not know which value must be selected for the commutation offset. This is mainly the case for axes with incremental or sine-cosine encoders, as these encoders cannot measure an absolute angular position for the rotor position.

NOTICE



Risk of unintentional start-up

Calling the function causes the motor to apply torque and start a movement. Take the following measures before using the function:

- ► Make sure that the settings of the controller parameters, at least those of the current/torque controller, are correct.
- ► Make sure that the motor shaft can rotate freely.

Parameters

Data type	Meaning
Enum DriveCommOffsetMeasurement-Modes (optional)	Defines the measuring method for the commutation offset.
DriveCommOffsetMeasurement-Modes.StaticMode	The static method causes the motor to lock into two angular positions by energizing it in a specific way. The motor shaft can move within the range of one pole pair of the motor.
DriveCommOffsetMeasurement-Modes.DynamicMode	The dynamic method readjusts the rotating field during the measurement of the resulting movement. This method leads to a significantly reduced movement of the motor shaft, but requires a more precise setting of the controllers.

Syntax

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

State management

ⓘ Info

Exclusive axis function

While the function is being executed, no other axis functions are possible for this axis. The next STX command is only executed when the measuring function has finished.

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	√	Inactive	Disabled
Enabled	X		Enabled
Running	X		Error: disabled
Suspended	X		Enabled
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3 Properties of MCAxis.Drive.Motor

The functionality provides the following properties:

– MotorDesign
– RatedCurrent
– MaxCurrent
– RatedSpeed
– RatedTorque
– Inductance
– Resistance

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

17.3.1 MotorDesign

Description

This property allows you to read the set motor design.

On some servo amplifiers, a different design can also be selected here in addition to the permanent-magnet synchronous motor (servo), for example a stepper motor.

Data type	Meaning
DriveMotorDesignTypes (optional)	Design of the motor.
DriveMotorDesignTypes.Unknown	The drive cannot be operated. Can only be read if initialization has failed.
DriveMotorDesignTypes.PMSM_Servo (default)	Permanent-magnet synchronous motor (servo motor). This is the predominant motor design in servo applications.
DriveMotorDesignTypes.Stepper_2Ph_CL	Closed-loop two-phase stepper motor. This design requires a position encoder for control.
DriveMotorDesignTypes.Stepper_2pH_OL	Open-loop two-phase stepper motor.
DriveMotorDesignTypes.Stepper_3pH_OL	Open-loop three-phase stepper motor.
DriveMotorDesignTypes.BLDC_3Ph	Electrically commutated three-phase DC motor.
DriveMotorDesignTypes.DC_HBridge	Permanent-magnet DC motor.
DriveMotorDesignTypes.ASM	Asynchronous motor.
DriveMotorDesignTypes.Reluctance	Reluctance motor.

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled

Initial state	Allowed	Final state
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.6 Inductance

Description

Data type	Meaning
Double	Reads the value of the inductance between two phases of the motor assigned to this axis in [mH].

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.7 Resistance

Description

Data type	Meaning
Double	Reads the value of the electrical resistance between two phases of the motor assigned to this axis in [Ω].

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.8 VoltageConstant

Description

Data type	Meaning
Double	Reads the value of the voltage constant of the motor assigned to this axis in [V_{eff}/ 1000 rpm].

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.9 TorqueConstant

Description

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

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.10 PolePairs

Description

Data type	Meaning
Double	Reads the value of the number of pole pairs of the motor assigned to this axis.

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.11 CommutationOffset

Description

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

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

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.13 ShaftPosition

Description

Data type	Meaning
Double	Reads the position of the motor shaft of the motor assigned to this axis in [0 … 360°].

Syntax

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

State management

The property can be read in any state.

17.3.14 Inertia

Description

Data type	Meaning
Double	Reads the value of the moment of inertia of the motor assigned to this axis in [kgcm²].

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running

Initial state	Allowed	Final state
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.15 TemperatureSensorType

Description

This property allows you to read and write the type of motor temperature sensor. The sensor type is not detected automatically, but must be specified according to the data sheet of the motor.

Data type	Meaning
Type DriveTemperatureSensorTypes (optional)	Type of motor temperature sensor.
DriveTemperatureSensorTypes.NoSensor (default)	No sensor connected. The sensor is not evaluated, i.e. motor temperature monitoring is not active.
DriveTemperatureSensorTypes.NCC	Temperature switch; The motor temperature cannot be read; only overtemperature monitoring is active.
DriveTemperatureSensorTypes.PTC	Temperature-dependent resistor with a non-linear, positively rising characteristic curve (PTC). The motor temperature cannot be read; only overtemperature monitoring is active.
DriveTemperatureSensorTypes.KTY83_110	Semiconductor sensor of type KTY83-110.
DriveTemperatureSensorTypes.KTY84_130	Semiconductor sensor of type KTY84-130.
DriveTemperatureSensorTypes.PT1000	Platinum resistance thermometer.
DriveTemperatureSensorTypes.MotorSpecific	For special motor/sensor combinations, e.g. LinMot motor.

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled

Initial state	Allowed	Final state
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.16 TemperatureSensorConnector

Description

This property allows you to read and write which connector the motor temperature sensor is connected to.

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

Data type	Meaning
Type DriveTemperatureSensorConnectors (optional)	Selection of the connector for the motor temperature sensor.
DriveTemperatureSensorConnectors.Motor (default)	The motor temperature sensor is connected to the motor connector. **NOTICE! This data type is required for a motor with a Hiperface DSL position encoder.**
DriveTemperatureSensorConnectors.Enc1	The motor temperature sensor is connected to the position encoder connector labeled "Enc1".
DriveTemperatureSensorConnectors.Enc2	The motor temperature sensor is connected to the position encoder connector labeled "Enc2".

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled

Initial state	Allowed	Final state
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.17 Temperature

Description

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

Syntax

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

State management

The property can be read in any state.

17.3.18 I2tOperatingMode

Description

Data type	Meaning
DriveI2tOperatingModes (optional)	Activation state of the motor I²t monitoring
DriveI2tOperatingModes.Disable (default)	I²t monitoring is deactivated.
DriveI2tOperatingModes.Enable	I²t monitoring is activated. For meaningful monitoring, the values for the thermal time constant [▶ 95] and the rated current [▶ 89] must match the motor used.

Syntax

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

State management

The property can be read in any state.

This property can also be accessed for writing, but under normal circumstances this does not make sense, as the motor cannot be changed while the machine is in operation.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled

Initial state	Allowed	Final state
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

17.3.19 I2tValue

Description

This value is only calculated if the I²t calculation is activated. For a meaningful value, the values for the thermal time constant [▶ 95] and the rated current [▶ 89] must match the motor 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 management

The property can be read in any state.

18 MCAxis.Drive.QuickStopDeceleration

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

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

18.1 Syntax for MCAxis.Drive.QuickStopDeceleration

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

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

18.2 Properties of MCAxis.Drive.QuickStopDeceleration

The functionality provides the following properties:

– QuickStopDeceleration

18.2.1 QuickStopDeceleration

Description

This property allows you to read or adjust the deceleration with which the axis is stopped when the `Power.Quickstop` command is triggered. This ramp is also applied in response to certain errors triggered in the servo amplifier.

The quick stop ramp is executed by the servo amplifier. While the quick stop ramp is being executed, the motion control function of the control unit no longer has any influence on the movement.

The servo amplifier stops the motor shaft in speed control with a linear deceleration ramp. The setting should ensure that the mechanics moved by the MCX axis can just about be stopped without damage.

Data type	Meaning
Double	Reads or writes the value for the deceleration of the quick stop ramp in [rpm/s].

Syntax

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

State management

The property can be read and written in any state.

19 MCAxis.Drive.TorqueShutOff

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

ⓘ Info

Required adjustment of parameters while torque shut-off is active

The **TorqueShutOff** functionality deliberately influences the control in the servo amplifier in an unusual way. In addition, the MCX axis is brought almost to a standstill by the tightening without a target position being reached. The MCX setpoint generation is not aware of this. It continues to output position setpoints that correspond to the `MovePtp.Start` command. As a result, the position following error and the difference between setpoint speed and actual speed can become higher than in normal positioning mode.

For this reason, it may be necessary to raise the parameters for Following error limit and Speed deviation window considerably, at least while the torque shut-off function is active.

ⓘ Info

Adjusting the feedforward while torque shut-off is active

The torque feedforward can also lead to unfavorable behavior of the axis during the sequence. It is recommended to deactivate this feedforward, at least while the torque shut-off function is active. To do so, set the parameter Torque feedforward scaling to `0`.

ⓘ Info

Parameter behavior with direction inversion

If an MCX axis is operated with direction inversion, this has no influence on the sign of the configuration parameters of the **torque shut-off** functionality.

A typical application is screwing caps onto bottles, especially plastic bottles: When a certain torque is reached (the cap is screwed on far enough), the movement must be stopped as quickly as possible, converting as much of the kinetic energy of the capping head as possible into electrical energy. This prevents the kinetic energy from passing into the screw connection and turning the cap further than desired. In this way, a more uniform capping result is achieved.

In the following, this sequence is explained using the example of screwing a cap onto a bottle. The sequence of a capping operation is shown in the following diagram, where the actual torque shut-off (*torque shut-off*) is executed within the gray frame.

There are 2 modes that relate to the behavior at the end of the sequence.

– In `HoldTorque` mode, a defined torque is applied at the end, with which the cap is held against the thread at standstill.
– In `HoldPosition` mode, the cap is controlled to the position that was 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 influence on the values in the parameters `TorqueTrigger`, `BrakeTorque`, `SpeedTrigger` and `HoldingTorque`. The units are Nm for the torques and rpm at the motor shaft for the speeds.

The diagram shows the qualitative curve of the torque (orange) and the speed (blue) of the motor shaft over time:

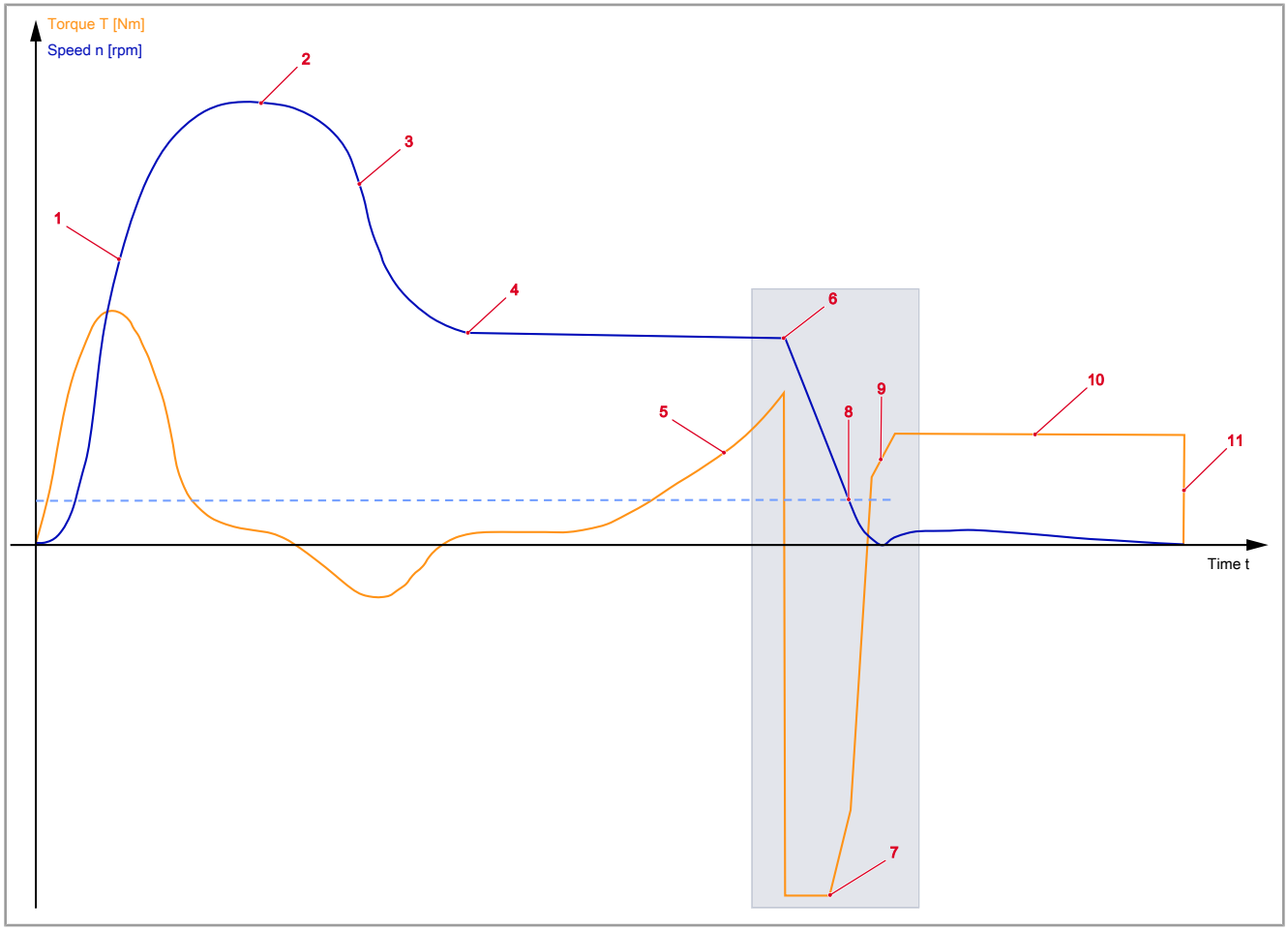

Fig. 1: MCX - TorqueShutOff

No.	Description
1	Acceleration phase to high speed. Since this acceleration can produce a higher (dynamic) torque than the shut-off, torque shut-off is not yet activated at this point.
2	Most of the thread is screwed at high speed.
3	Braking to low speed. In this way, the energy stored in the inert mass of the capping head and the motor is kept as low as possible. This energy depends directly on the speed.
4	Low speed reached: Torque shut-off is activated. This is where the actual sequence within the MCX function starts.
5	The cap approaches the screwing position: The torque increases.
6	Torque threshold (`TorqueTrigger`) reached: The `TorqueShutOff` function limits the setpoint speed to 0. This causes the speed controller to reduce the speed as much as possible at the set torque limit. In support, a "braking torque" (`BrakeTorque`) is written once to the integral component of the speed controller in order to bring the control into the braking process as quickly as possible. These two actions result in the fastest possible braking against the inertia of the capping head.
7	The difference between setpoint speed and actual speed decreases: The torque leaves the limitation.

No.	Description	
8	When the specified speed threshold (`SpeedTrigger`) is reached, the speed limit is removed to prevent the cap from turning back. Steps 9 and 10 depend on the mode:	
	Mode `HoldPosition`	**Mode** `HoldTorque`
9	A positive torque value (`HoldingTorque`) is written once to the integral component of the speed controller. This influences the control in such a way that a transition to position control that is as jerk-free as possible is ensured.	The torque limit is set to the value of `HoldingTorque`.
10	The setpoint position is set to the position the shaft had at the speed threshold: The cap is held at this position in position control.	The torque limit remains active until the torque shut-off function is deactivated. As a result, the cap may continue to move very slowly with a defined torque until the final position is reached.
11	At the end of the capping operation, the position controller is set to the current position. This is necessary because the torque shut-off function has manipulated the controllers "below" the MCX setpoint generation, with the effect that the actual value of the position has not followed the setpoint as usual. The `MCAxis.State.Transitions.UserPositionReset()` command allows you to set the setpoint position to the value of the actual position and thus set position control to a defined initial position. Alternatively, you can disable the axis with the `MCAxis.Power.Disable()` command. When it is enabled again, the setpoint position is also preset with the actual position, so that no jerk occurs. In addition, in this case all internal controllers start without an accumulated integral sum.	
	Finally, the torque shut-off function is deactivated again.	

19.1 Syntax for MCAxis.Drive.TorqueShutOff

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Drive.TorqueShutOff.<function>(optional <parameter>);
```

19.2 Functions of MCAxis.Drive.TorqueShutOff

The functionality provides the following functions:

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

19.2.1 Configure()

This function passes the information that is suitable for your application to the **Drive.TorqueShutOff** functionality.

Parameters

Parameter	Data type	Meaning
Mode	DriveTorqueShutOffModes	Defines the control mode at the end of the sequence.
	DriveTorqueShutOff-Modes.HoldPosition	At the end of the sequence, controls to the position the axis had at the speed threshold.
	DriveTorqueShutOff-Modes.HoldTorque	Applies the value of `HoldingTorque` at the end of the sequence. Using the example of a capping operation: torque with which the axis "presses" against the thread.
TorqueTrig-ger	Double	Torque value above which torque shut-off is to become active, in [Nm at the motor shaft]. This value is an absolute value and is always used as a positive value.
BrakeTorque	Double	Torque with which the integral component of the speed controller is to be preset during rapid braking, in [Nm at the motor shaft]. This value must be negative if braking is to be supported very strongly.
SpeedTrig-ger	Double	Speed threshold below which the braking process is aborted 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 used as a positive value.
Holding-Torque	Double	– In `HoldPosition` mode: Torque with which the integral component of the speed controller is preset in order to obtain a transition to position control after braking that is as jerk-free as possible. – In `HoldTorque` mode: Torque that the axis continues to apply after braking. Using the example of a capping operation: Torque with which the axis continues to "press" against the thread, in [Nm at the motor shaft]. This value is an absolute value and is always used as a positive value.

Syntax

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

State management

The function can be called in any state.

19.2.2 ReadConfiguration()

The function reads the currently effective configuration of the **Drive.TorqueShutOff** functionality.

Parameters

The function has no parameters. The return value is output in a structure of type `DriveTorqueShutOff-Configuration`. It has the following elements:

Parameter	Data type	Meaning
Mode	DriveTorqueShutOffModes	Defines the control mode at the end of the sequence.
	DriveTorqueShutOff-Modes.HoldPosition	At the end of the sequence, controls to the position the axis had at the speed threshold.
	DriveTorqueShutOff-Modes.HoldTorque	Applies the value of `HoldingTorque` at the end of the sequence. Using the example of a capping operation: torque with which the axis "presses" against the thread.
TorqueTrig-ger	Double	Torque value above which torque shut-off is to become active, in [Nm at the motor shaft]. This value is an absolute value and is always used as a positive value.
BrakeTorque	Double	Torque with which the integral component of the speed controller is to be preset during rapid braking, in [Nm at the motor shaft]. This value must be negative if braking is to be supported very strongly.
SpeedTrig-ger	Double	Speed threshold below which the braking process is aborted 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 used as a positive value.
Holding-Torque	Double	– In `HoldPosition` mode: Torque with which the integral component of the speed controller is preset in order to obtain a transition to position control after braking that is as jerk-free as possible. – In `HoldTorque` mode: Applies the value of `HoldingTorque` at the end of the sequence. Using the example of a capping operation: torque with which the axis "presses" against the thread. This value is an absolute value and is always used as a positive value.

Syntax

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

State management

The function can be read in any state.

19.2.3 Activate()

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

Parameters

The function has no parameters.

Syntax

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

State management

The function can be called in any state.

19.2.4 Deactivate()

This function deactivates the **Drive.TorqueShutOff** functionality. No more functions of the function are executed. All torque and speed limits are reset to the originally parameterized value.

Parameters

The function has no parameters.

Syntax

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

State management

The function can be called in any state.

19.3 Properties of MCAxis.Drive.TorqueShutOff

The functionality provides the following properties:

– State

19.3.1 State

Description

Data type	Meaning
DriveTorqueShutOffStates (optional)	Reads the current status of the **Drive.TorqueShutOff** functionality in the sequence.
DriveTorqueShutOffStates.Disabled	The function is not active.
DriveTorqueShutOffStates.WaitForTorque	The function has been activated. The torque at the motor shaft is still lower than the value in `TorqueTrigger`.
DriveTorqueShutOffStates.WaitForSpeed	`TorqueTrigger` has been exceeded. Braking has been initiated. The speed has not yet fallen below the value in `SpeedTrigger`.
DriveTorqueShutOffStates.Finished	The sequence has finished. Depending on the configured mode, the position or the torque is controlled.

Syntax

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

State management

The property can be read in any state.

19.4 Example task MCAxis.Drive.TorqueShutOff

Here you will find an example task for a single axis that represents the sequence in the diagram 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();
        When 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 become stable
    Resol_3000.Drive.TorqueShutOff.Deactivate();

    bStartDA := False;

End_Loop;
End_Task;
```

20 MCAxis.Info

The **MCAxis.Info** functionality allows you to identify the axis objects of the motion control.

This is information that the user has defined in the hardware manager and motion setup in the project.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

20.1 Syntax for MCAxis.Info

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Info.<property>;
```

20.2 Properties of the functionality

The functionality provides the following properties:

– Name
– ObjectID
– ClassID

20.2.1 Name

Description

Data type	Meaning
String	Queries the name of the single axis 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 management

The property can be read in any state.

20.2.2 ObjectID

Description

Data type	Meaning
Integer	Queries the controller-internal number of the path group. The property is normally only required in connection with applications that use the functions of the MC manager. A possible application is described in the corresponding section MCManager (MCMgr) [▶ 402].

Syntax

```
ObjectIdentifier := AX1.Info.ObjectID;
```

State management

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. The property is normally only required in connection with applications that use the functions of the MC manager. A possible application is described in the corresponding section MCManager (MCMgr) [▶ 402].

Syntax

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

State management

The property can be read in any state.

20.3 Example task MCAxis.Info

Here you will find an example task for a single axis that contains various Motion API functions:

After the example task has been executed, the trace shows for all axes whether or not they are in the **running** state. The axes are listed with 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 by 1 do
        TraceRunningState(mcMgr.AxisList[nAxisIndex]);
    end_for;
end_task;
```

21 MCAxis.IsInTargetWindow

The **MCAxis.IsInTargetWindow** functionality allows you to query 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 the motion setup or via the `MCAxis.MovePtp.Start()` function. As soon as the axis object has reached its target window, the property is `IsInTargetWindow = true`. When a new positioning movement starts, the property is `IsInTargetWindow = false`.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

21.1 Syntax for MCAxis.IsInTargetWindow

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.IsInTargetWindow
```

21.2 Properties of MCAxis.IsInTargetWindow

`IsInTargetWindow` is a property of the axis object.

21.2.1 IsInTargetWindow

Description

Data type	Meaning
Bool	Queries whether the actual position of the axis object is within the target window.

Syntax

```
When AX1.IsInTargetWindow Continue;
```

State management

The property can be queried in any state.

21.3 Example task MCAxis.IsInTargetWindow

Here you will find an example task for a single axis that contains various Motion API functions:

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

Task Sample2 Autorun

loop
    when run Continue;

    //Switch 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
switch-on position.
    When AX1.MoveHome.IsReferenceSet Continue;

    //Position the axis absolutely to position 10000 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
);
    When AX1.IsInTargetWindow Continue;

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

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

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

    //If the user has meanwhile cleared the auxiliary flag in the setup,
disable the axis and wait for it to be set again.
    //Otherwise, repeat reversing the axis.
    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

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

The **MCAxis.Mechanism.AxisMode** functionality allows you to read the mode of an MCX axis that was configured in the hardware manager of JetSym.

The mode determines which functionality the axis has with regard to the various MCX functions and how the axis can be used. Available axis modes:

– **InternalAxis**

  An internal axis is a real axis that is connected to the system bus of the control unit and can potentially move mass thanks to its motor and servo amplifier.

  For an axis with this mode, all settings required for real operation can be made.

– **VirtualAxis**

  A virtual axis acts as an auxiliary axis that behaves like a real axis from the point of view of motion control, but cannot perform any real movement. The actual position of this axis is always equal to the setpoint position. The axis function is only calculated in the control unit.

  Due to the restricted function, only mechanical parameters can be set.

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

– **ShadowAxis**

  A shadow axis is used to cascade technology groups. The shadow axis is "attached" to another (internal or virtual) axis, i.e. it adopts all parameters and the behavior of its so-called data axis. Above all, the position of the data axis serves as the master axis position for further technology groups. In this way, for example, the position of a follower axis in one technology group can be used as the master axis of another technology group (cascading).

  A shadow axis must always be used as a master axis in a technology group.

  Due to the restricted function, only the units of the mechanical parameters can be changed.

– **ExternalAxis**

  An external axis is an MCX axis that only reads in the position of a position encoder from the machine. The movement of this axis is not influenced by the MCX control. It can be evaluated in the STX program as the current position of this machine part or used as the master axis position for a technology group.

  The encoder evaluated in the external axis must be connected to the second encoder interface of an internal axis. This internal axis must be parameterized in such a way that the position of this encoder is read in correctly and made known to the MCX control for external axes.

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

  Due to the restricted function, only mechanical parameters can be set.

– **SimulatedAxis**

  A simulated axis is an axis that behaves like an internal axis, but whose servo amplifier and motor are simulated. With a simulated 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 part of the path movement.

You can only change the configured mode in the hardware manager of JetSym. For the change to take effect, you must transfer the configuration of the project to the control unit again.

22.1 Syntax for MCAxis.Mechanism.AxisMode

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.AxisMode
```

22.2 Properties of MCAxis.Mechanism.AxisMode

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

22.2.1 AxisMode

A detailed description of the axis modes can be found in *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 section MCAxis.Mechanism.AxisMode [▶ 114].

Syntax

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

State management

The property can be read in any state.

23 MCAxis.Mechanism.AxisType

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

The control unit must know the mechanical design of an axis so that the units and the kinematics are calculated correctly in the MCX. This property defines whether the axis is endless or limited, and whether linear or non-linear mechanics are moved by the axis.

23.1 Syntax for MCAxis.Mechanism.AxisType

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.AxisType
```

23.2 Properties of MCAxis.Mechanism.AxisType

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

23.2.1 AxisType

This property allows you to read or set the mechanical type of the axis.

A detailed description of the mechanical axis types can be found in the chapter Description of the axis types [▶ 117].

Data type	Meaning
AxisTypes (optional)	Defines the mechanical type of the axis.
Rotatory	The axis configuration is rotary/normal.
RotatoryModulo	The axis configuration is rotary/modulo.
RotatoryModuloUniPos	The axis configuration is rotary/unidirectional/modulo positive.
RotatoryModuloUniNeg	The axis configuration is rotary/unidirectional/modulo negative.
Linear	The axis configuration is linear/normal.
LinearModulo	The axis configuration is linear/modulo.
LinearModuloUniPos	The axis configuration is linear/unidirectional/modulo positive.
LinearModuloUniNeg	The axis configuration is linear/unidirectional/modulo negative.
Crank	The axis is configured as a non-linear crank axis.
Piston	The axis is configured as a non-linear piston axis.

Syntax

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

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

23.3 Description of the axis types

The mechanical axis types can be divided into 4 basic types with subtypes:

– Rotary
– Linear
– Crank
– Piston

Whether the axis is a rotary axis or a linear axis is already defined in the motion setup of JetSym (page "Axis parameters").

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

23.3.1 Rotary axis

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

A gearbox with a gear ratio may be installed, as a result of which the number of motor revolutions does not match the number of mechanical revolutions. The motion parameters (position, speed etc.) always refer to the movement of the load.

Rotary/normal

This axis type is used if the movement of the load takes place between two limits. As a result, the load is moved in both directions. A maximum travel range can be defined for the load.

Rotary/modulo

With this axis type, there is no limiting travel range. The travel range only defines the so-called modulo travel range.

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

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

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. with a rotary indexing table).

Rotary/modulo/unidirectional

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

Example:

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

23.3.2 Linear axis

With a linear axis, the connected load is moved in a straight line. With a rotary motor, mechanics are required that convert the rotating movement of the motor axis (including any gear ratio) into a linear movement 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, speed etc.) always refer to the movement of the mechanics.

Linear/normal

This axis type is used if the movement of the load takes place between two limits. As a result, the load is moved in both directions. A maximum travel range can be defined for the load.

Linear/modulo

With this axis type, there is no limiting travel range. The travel range only defines the so-called modulo travel range.

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

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

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. with a conveyor belt).

Linear/modulo/unidirectional

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

Example:

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

23.3.3 Non-linear eccentric axis

This axis type is used if a linear movement of the load is produced by a (rotating) eccentric.

The travel range corresponds to twice the distance between the eccentric and the axis center point of the rotating mechanics. The kinematic conversion in the MCX allows you to program in linear units of measurement relative to the load.

To use this axis type, the following must be communicated to the MCX:

– 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 formula:

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

Where:

`Radius` is the distance between the eccentric and the axis center point of the rotating mechanics.

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

`Offset` is the shift of the linear position.

The `InvertDirection` parameter determines the sign in front of the bracket in the formula.

23.3.4 Non-linear piston axis

This axis type is used if a linear movement of the load is produced by a (rotating) eccentric (crankshaft) connected to a piston rod.

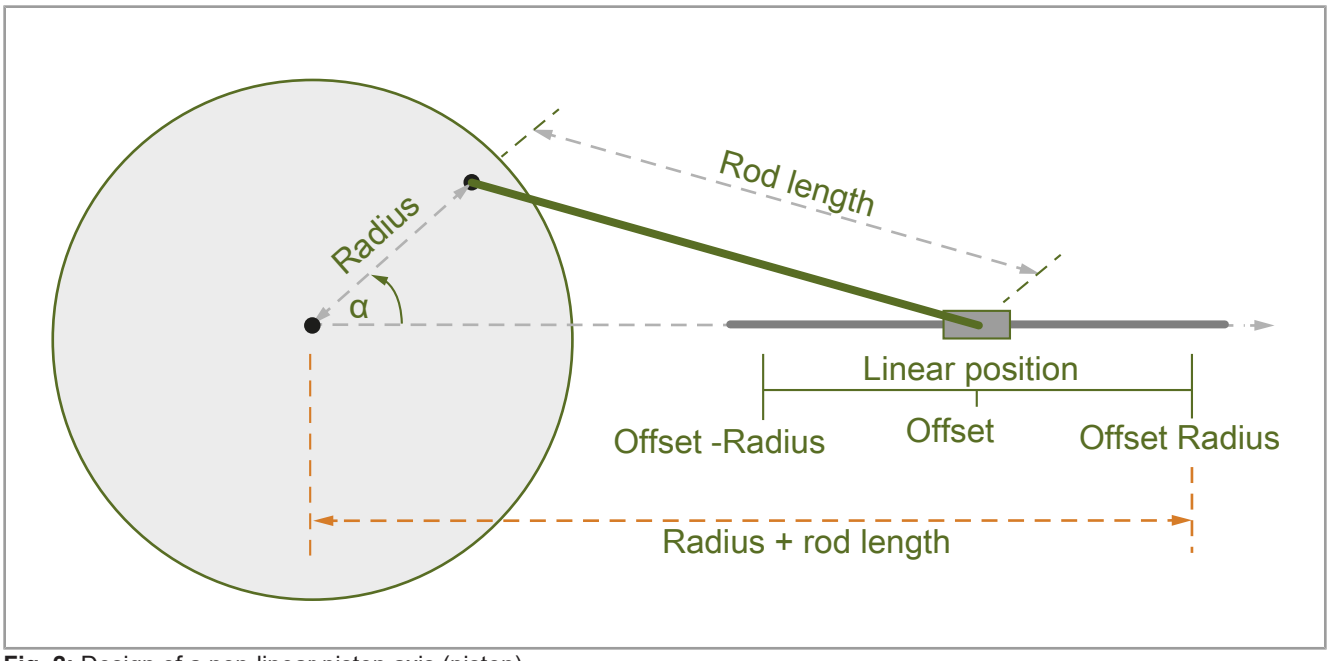

Fig. 2: Design of a non-linear piston axis (piston)

The travel range corresponds to twice the distance between the eccentric and the axis center point of the rotating mechanics (radius). The kinematic conversion in the MCX allows you to program in linear units of measurement relative to the load.

To use this axis type, the following must be communicated to the MCX:

– 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 formula:

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

  Where:

  Radius is the distance between the eccentric and the axis center point of the rotating mechanics.

  RodLength is the length of the piston rod.

  α is the angle of rotation of the eccentric disc.

  Offset is the shift of the linear position.

  The `InvertDirection` parameter determines the sign in front of the bracket in the formula.

23.4 Example task for MCAxis.Mechanism.AxisType

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample_AxisType Autorun
    …
If AX2.State.IsDisabled then;
    AX2.Mechanism.Kinematics.Piston.Configure(Offset,InvertDir,Radius,RodLen
gth);
    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

The **MCAxis.Mechanism.GearRatioLoad** and **MCAxis.Mechanism.GearRatioMotor** functionalities allow you to read and adjust the ratio of the gearbox used.

ⓘ Info

If one of the two properties is changed for an axis that has already been homed, the homing is lost. Homing must be repeated.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. 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 functionality in the JetSym editor is via IntelliSense, as follows:

```
<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 in relation to the output speed. Only integer values are permitted.

Example

For a gear ratio of i = 3.75, select the value 375 for `GearRatioMotor` and the value 100 for `GearRatioLoad`.

Syntax

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

State management

The property can be read in any state.

The following table applies to writing the gear ratio values:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

24.2.2 GearRatioLoad

Description

Data type	Meaning
Double	Specifies the load-side shaft speed of the gearbox in relation to the input speed. Only integer values are permitted.

Example

For a gear ratio of i = 3.75, select the value 375 for `GearRatioMotor` and the value 100 for `GearRatioLoad`.

Syntax

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

State management

The property can be read in any state.

The following table applies to writing the gear ratio values:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

25 MCAxis.Mechanism.Kinematics

The **MCAxis.Mechanism.Kinematics** functionality allows you to read and adjust the mechanical design of a non-linear eccentric axis (crank [▶ 119]) or non-linear piston axis (piston [▶ 119]).

The control unit must know the mechanical design of an axis so that the units and the kinematics are calculated correctly in the MCX. The kinematic data must be made known to the MCX before the axis type is defined. After the kinematics have been defined, you must switch the axis type of the axis using the Motion API command MCAxis.Mechanism.AxisType.

A detailed description of the kinematic function of these two axis types can be found under the command MCAxis.Mechanism.AxisType [▶ 116].

25.1 Syntax for MCAxis.Mechanism.Kinematics

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.Kinematics.<function or value access>(optional <param-
eter 1, parameter 2, parameter n>);
```

25.2 Functions of MCAxis.Mechanism.Kinematics

The functionality provides 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 to the MCX in one command. Using these properties, the MCX can calculate the kinematic transformation. This allows you to program the desired linear movement of the load in linear units, although the driving motor is rotary.

The properties of the function can only be accessed for reading.

Parameters

Parameter / Properties	Data type	Meaning
Offset	Double (optional)	Shifts the linear position.
InvertDirection	Modes (optional)	Defines the counting direction of the linear position.
Radius	Double (optional)	Defines the distance between the axis center point and the eccentric (or the attachment point of the piston rod).
RodLength	Double (optional)	Piston only: 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 management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

25.3 Properties of MCAxis.Mechanism.Kinematics

The functionality provides the following properties:

– Crank.Radius
– Crank.Offset
– Crank.InvertDirection
– Piston.Radius
– Piston.RodLength
– Piston.Offset
– Piston.InvertDirection

25.3.1 .Offset / .InvertDirection / .Radius / .RodLength

Description

The `Offset`, `InvertDirection`, `Radius` and `RodLength` properties allow you to read the currently valid kinematic parameters of the eccentric or piston axis.

Property	Data type	Meaning
Offset	Double (optional)	Shifts 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 axis center point and the eccentric (or the attachment point of the piston rod).
RodLength	Double (optional)	Piston only: 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;
EccentricDirection := AX1.Mechanism.Kinematics.Crank.InvertDirection
```

State management

The property can be read in any state.

To write the properties, use the `Configure()` function.

25.4 Example task MCAxis.Mechanism.Kinematics

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample_AxisType Autorun

    …

    If AX2.State.IsDisabled then
        AX2.Mechanism.Kinematics.Piston.Configure(Offset,InvertDir,Radius,Ro
dLength);
        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** functionality allows you to pass the mechanical conditions of the MCX axis in the machine to the MCX control. These are used as limit values for motion programming.

If the values for the travel range limits are exceeded by the programming of a movement, an error is output and the movement is not started.

For a modulo axis, the position limits define the modulo range. If the axis exceeds the value of `TravelLimitMax`, the position continues at the value of `TravelLimitMin` without stopping. Analogously: If the axis falls below the value of `TravelLimitMin`, the position continues at the value of `TravelLimitMax` without stopping.

When the motion profile is calculated, the maximum values for speed, acceleration and, in the case of jerk-limited motion types, also for jerk are taken into account. If the motion command results in higher values for speed, acceleration and jerk, these are automatically limited to the maximum values without an error message.

26.1 Syntax for MCAxis.Mechanism.Limitations

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.Limitations.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

26.2 Functions of MCAxis.Mechanism.Limitations

The functionality provides the following functions:

– SetTravelLimits()

26.2.1 SetTravelLimits()

This function allows you to specify the travel range in load position units. This is done by specifying the smallest and the largest position value in the travel range relative to the reference point.

For a modulo axis, these two values define the modulo range. If the axis exceeds the value of `TravelLimitMax`, the position continues at the value of `TravelLimitMin` without stopping. Analogously: If the axis falls below the value of `TravelLimitMin`, the position continues at the value of `TravelLimitMax` without stopping.

ⓘ Info

Loss of homing by executing SetTravelLimits

After this function has been executed, the axis is no longer homed. A previously performed homing must be repeated. This does not apply to an axis with an absolute encoder whose homing was carried out via a reference shift (see SetAbsEncoderReference() [▶ 155]).

To read the values for the travel range/modulo range, use the `TravelLimitMin` and `TravelLimitMax` properties.

Parameters

Parameter	Data type	Meaning
TravelLimitMin	Double (optional)	Smallest position value in the travel range/modulo range; absolute value relative to the reference point.

Parameter	Data type	Meaning
TravelLimitMax	Double (optional)	Largest position value in the travel range/modulo range; absolute value relative to the reference point.

Syntax

```
AX1.Mechanism.Limitations.SetTravelLimit(0.0, 360.0);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

26.3 Properties of MCAxis.Mechanism.Limitations

The functionality provides 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 modulo range (see function SetTravelLimits() [▶ 126]).

Syntax

```
TravelRangeLimitMin := AX1.Mechanism.Limitations.TravelLimitMin;
AX1.Mechanism.Limitations.TravelLimitMin := -100.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

26.3.2 TravelLimitMax

Description

Data type	Meaning
Double (optional)	Reads or writes the larger value for the travel or modulo range (see function SetTravelLimits() [▶ 126]).

Syntax

```
TravelRangeLimitMax := AX1.Mechanism.Limitations.TravelLimitMax;
AX1.Mechanism.Limitations.TravelLimitMax := 1250.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

26.3.3 MaxSpeed

Description

Data type	Meaning
Double (optional)	Reads or writes the value for the maximum speed at which the load can or should be moved. This value can be determined, for example, by the function of the machine or by the maximum data of a threaded spindle or other components of the drive system.

Syntax

```
MaximumSpeed := AX1.Mechanism.Limitations.MaxSpeed;
AX1.Mechanism.Limitations.MaxSpeed := 1.5; // m/s
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

26.3.4 MaxAcceleration

Description

Data type	Meaning
Double (optional)	Reads or writes the value for the maximum acceleration at which the load can or should be moved. This value can be determined, for example, by the function of the machine or by the maximum data of a threaded spindle or other components of the drive system. The monitoring applies this value to both the acceleration and the deceleration phase of a movement.

Syntax

```
MaximumAcceleration := AX1.Mechanism.Limitations.MaxAcceleration;
AX1.Mechanism.Limitations.MaxAcceleration := 15000; // °/s²
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

26.3.5 MaxJerk

Description

Data type	Meaning
Double (optional)	Reads or writes the value for the maximum jerk with which the load can or should be moved. This value can be determined, for example, by the function of the machine or by the maximum data of a threaded spindle or other components of the drive system.

Syntax

```
MaximumJerk := AX1.Mechanism.Limitations.MaxJerk;
AX1.Mechanism.Limitations.MaxJerk := 20000.0; // m/s³
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

27 MCAxis.Mechanism.LinearRotationRatio

The **MCAxis.Mechanism.LinearRotationRatio** functionality allows you to read and adjust the ratio of the mechanical components that convert a rotating motor movement into a linear movement (e.g. toothed belt, spindle, roller feed). This functionality only affects axes configured as linear.

The ratio is taken into account in such a way that these components take effect at the gearbox output. This means that if no gearbox is installed, a ratio of 1:1 must be parameterized.

27.1 Syntax for MCAxis.Mechanism.LinearRotationRatio

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.LinearRotationRatio
```

27.2 Properties of MCAxis.Mechanism.LinearRotationRatio

The functionality provides the following properties:

The **LinearRotationRatio** functionality is a property of the **Mechanism** function.

27.2.1 LinearRotationRatio

Description

Data type	Meaning
Double	Specifies the conversion of the rotary movement into a linear movement on the gearbox output side, i.e. the distance the load is to move when the gearbox output shaft performs one revolution. Example: For a ball screw with a pitch of 10 mm, the value to be entered is 10 mm/rev.

Syntax

```
AX1.Mechanism.LinearRotationRatio:= 10.0;
LinRotConversion := AX1.Mechanism.LinearRotationRatio;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	–	Enabled
Running	–	Running
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

28 MCAxis.Mechanism.Monitoring

The **MCAxis.Mechanism.Monitoring** functionality allows you to query the state of the limit switch inputs and of the reference input of the servo amplifier of the MCX axis.

The signals are displayed taking the set polarity into account. For example, an active hardware limit switch on an axis that is in movement leads to an error.

You set the polarity of the signal evaluation and the mode of the inputs via the properties of the MCAxis.Drive.DigitalInput [▶ 72] functionality.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

28.1 Syntax for MCAxis.Mechanism.Monitoring

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.Monitoring.<value access>;
```

28.2 Properties of MCAxis.Mechanism.Monitoring

The functionality provides 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 of the servo amplifier. If the status is "active", the reference switch is detected during homing of the MCX axis and a corresponding response is executed, if applicable.

Syntax

```
ReferenceSwitchStatus := AX1.Mechanism.Monitoring.IsReferenceSwitchActive;
```

State management

The property can be read in any state.

The property cannot be written, so the signal function cannot be activated by software.

28.2.2 IsHWLimitSwitchMinActive

Description

This property allows you to read the status of the hardware limit switch in the negative travel direction that is connected to the corresponding digital input of the servo amplifier.

If the status is *active*, an error is output and the axis is stopped and disabled. Positioning of the axis is only possible in the opposite direction.

Data type	Meaning
Bool	Evaluates the hardware limit switch in the negative travel direction as *inactive* or *active*.

Syntax

```
NegLimitSwitchStatus := AX1.Mechanism.Monitoring.IsHWLimitSwitchMinActive;
```

State management

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 in the positive travel direction that is connected to the corresponding digital input of the servo amplifier.

If the status is *active*, an error is output and the axis is stopped and disabled. Positioning of the axis is only possible in the opposite direction.

Data type	Meaning
Bool	Evaluates the hardware limit switch in the positive travel direction as *inactive* or *active*.

Syntax

```
PosLimitSwitchStatus := AX1.Mechanism.Monitoring.IsHWLimitSwitchMaxActive;
```

State management

The property can be read in any state.

29 MCAxis.Mechanism.Slope

The **MCAxis.Mechanism.Slope** functionality allows you to read or adjust the calculated properties of the current or next movement (Config.RampBreakMode [▶ 141]).

A positioning movement consists of the following phases:

1. Acceleration ramp
2. Travel at constant speed

   This phase only occurs if the speed `Speed` can be reached with the specified MovePtp parameters. Otherwise, the acceleration ramp changes directly into the deceleration ramp (triangular mode).
3. Deceleration ramp

In addition, there can also be no movement if the axis is at its target and no further positioning has been started yet.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

29.1 Syntax for MCAxis.Mechanism.Slope

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.Slope.<value access>
```

29.2 Properties of MCAxis.Mechanism.Slope

The functionality provides the following properties:

– SlopeState
– IsAccelerating
– IsAtConstantSpeed
– IsDecelerating
– IsStopped
– SlopeType
– ActualMaxSpeed
– ActualMaxAcceleration
– ActualMaxDeceleration
– SpeedSetpoint
– AccelerationSetpoint
– Speed
– Acceleration
– Deceleration
– Config.RampBreakMode

29.2.1 SlopeState

Description

Data type	Meaning
AxisSlopeStates	Reads which phase of the movement the calculated setpoint of the MCX axis is in. This allows you, for example, to activate an action in the machine only after the MCX axis has left the acceleration phase.
AxisSlopeStates.Accelerating	Axis is in the acceleration ramp.
AxisSlopeStates.AtConstandSpeed	Axis is moving at constant speed.
AxisSlopeStates.Decelerating	Axis is in the deceleration ramp.
AxisSlopeStates.Stopped	Axis is at standstill.

Syntax

```
RampStatus := AX1.Mechanism.Slope.SlopeState;
```

State management

The property can be read in any state.

29.2.2 IsAccerlerating

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the MCX axis is in the acceleration ramp.

Syntax

```
If AX1.Mechanism.Slope.IsAccelerating then …
```

State management

The property can be queried in any state.

29.2.3 IsAtConstantSpeed

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the MCX axis is in the constant speed phase.

Syntax

```
If AX1.Mechanism.Slope.IsAtConstantSpeed then …
```

State management

The property can be queried in any state.

29.2.4 IsDecelerating

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the MCX axis is in the deceleration ramp.

Syntax

```
If AX1.Mechanism.Slope.IsDecelerating then …
```

State management

The property can be queried in any state.

29.2.5 IsStopped

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the MCX axis is in the standstill phase.

Syntax

```
If AX1.Mechanism.Slope.IsStopped then …
```

State management

The property can be queried in any state.

29.2.6 SlopeType

Description

The `SlopeType` property allows you to read or adjust the ramp shape in the acceleration and deceleration phase. **NOTICE! This setting applies to all positioning movements of the MCX axis until the value is changed.**

Data type	Meaning
AxisSlopeTypes	Shape of the acceleration and deceleration ramp.
AxisSlopeTypes.Sine	This setting calculates a movement that generates relatively little jerk, i.e. the movement is smoother. Since the acceleration value passed to the motion command is only reached in the middle of the ramp, ramp duration and length are greater than with the `linear` setting. In the acceleration phase, the movement is sinusoidal, and in the deceleration phase, it is sin²-shaped. The movement is jerk-limited.
AxisSlopeTypes.Linear	Compared with the `sine` setting, this setting enables a movement with the lowest possible maximum acceleration for the same ramp time (and length). In return, the jerk is infinite at the transition from standstill to the acceleration phase and from the acceleration phase to the phase with maximum speed. In the acceleration phase, the movement is constant, and in the deceleration phase, it is sinusoidal. The movement takes place without jerk limitation.

Syntax

```
RampType := AX1.Mechanism.Slope.SlopeType;
AX1.Mechanism.Slope.SlopeType := AxisSlopeTypes.Linear;
```

State management

The property can be read and written in any state.

29.2.7 ActualMaxSpeed

Description

Data type	Meaning
Double	Reads the value of the maximum speed currently programmed in the setpoint generator of the MCX axis for the current or next movement. This value can be changed by specifying a new speed value in the commands `MCAxis.MoveVelocity.Start()`, `MCAxis.MoveVelocity.NewSpeed()`, `MCAxis.MovePtp.Start()`, `MCAxis.MovePtp.NewSpeed()`. If the parameter is not specified in these commands, the value readable with `ActualMaxSpeed` is used in the movement.

Syntax

```
MaxSpeed := AX1.Mechanism.Slope.ActualMaxSpeed;
```

State management

The property can be read in any state.

29.2.8 ActualMaxAcceleration

Description

Data type	Meaning
Double	Reads the value of the maximum acceleration last passed to the setpoint generator of the MCX axis. This value can be changed by specifying a new acceleration value in the commands `MCAxis.MoveVelocity.Start()`, `MCAxis.MoveVelocity.NewAcceleration()`, `MCAxis.MovePtp.Start()`, `MCAxis.MovePtp.NewAcceleration()`. If the parameter is not specified in these commands, the value readable with `ActualMaxAcceleration` is used in the movement.

Syntax

```
MaxAcceleration := AX1.Mechanism.Slope.ActualMaxAcceleration;
```

State management

The property can be read in any state.

29.2.9 ActualMaxDeceleration

Description

Data type	Meaning
Double	Reads the value of the maximum acceleration last passed to the setpoint generator of the MCX axis. This value can be changed by specifying a new deceleration value in the commands `MCAxis.MoveVelocity.Start()`, `MCAxis.MoveVelocity.NewDeceleration()`, `MCAxis.MovePtp.Start()`, `MCAxis.MovePtp.NewDeceleration()`. If the parameter is not specified in these commands, the value readable with `ActualMaxDeceleration` is used in the movement.

Syntax

```
MaxDeceleration := AX1.Mechanism.Slope.ActualMaxDeceleration;
```

State management

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 effective in the setpoint generator of the MCX axis. The sign of the value indicates acceleration or deceleration.

Syntax

```
ActAccelerationSetpoint := AX1.Mechanism.Slope.AccelerationSetpoint;
```

State management

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 effective in the setpoint generator of the MCX axis. The sign of the value depends on the travel direction.

Syntax

```
ActSpeedSetpoint := AX1.Mechanism.Slope.SpeedSetpoint;
```

State management

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 for the position setpoint generation of a positioning profile. This value is normally passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` or `MovePtp.NewSpeed()` and `MoveVelocity.NewSpeed()`, where a change is applied to a running movement. This is different here: Regardless of the state in which the change of the value for `Speed` is passed, the new value only takes effect in the next movement.

Syntax

```
ProfileSpeed := AX1.Mechanism.Slope.Speed;
AX1.Mechanism.Slope.Speed := 3000.0;
```

State management

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 for the position setpoint generation of a positioning profile. This value is normally passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` or `MovePtp.NewAcceleration()` and `MoveVelocity.NewAcceleration()`. Regardless of the state in which the change of the value for `Acceleration` is passed, the new value only takes effect in the next movement.

Syntax

```
ProfileAcceleration := AX1.Mechanism.Slope.Acceleration;
AX1.Mechanism.Slope.Acceleration := 5500.0;
```

State management

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 deceleration used for the position setpoint generation of a positioning profile. This value is normally passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` or `MovePtp.NewDeceleration()` and `MoveVelocity.NewDeceleration()`. Regardless of the state in which the change of the value for `Deceleration` is passed, the new value only takes effect in the next movement.

Syntax

```
ProfileDeceleration := AX1.Mechanism.Slope.Deceleration;
AX1.Mechanism.Slope.Acceleration := 12000.0;
```

State management

The property can be read and written in any state.

29.2.15 Config.RampBreakMode

Description

Deceleration ramp with MovePtp.NewDeceleration()

If the deceleration is not passed with the `AX1.MovePtp.Start()` command but with `AX1.MovePtp.NewDeceleration()`, positioning is always executed according to the ramp configuration `immediate`.

Data type	Meaning
MCAxisRampBreakModes	Reads or writes the behavior of the setpoint generation when a new `MovePtp.Start()` command is issued while the movement is in the deceleration ramp. The value after switching on is `None`. This is a static configuration that remains active until it is changed.
MCAxisRampBreakModes.Immediate	1. The travel distance to standstill is sufficient: The deceleration ramp is aborted with the newly passed maximum deceleration and the second target position is approached directly. 2. The travel distance to standstill is not sufficient: The first target position is approached. Positioning to the second target position follows directly.
MCAxisRampBreakModes.None	The first target position is approached. Positioning to the second target position follows directly.

Syntax

```
RampConfiguration := AX1.Mechanism.Slope.Config.RampBreakMode;
AX1.Mechanism.Slope.Config.RampBreakMode := MCAxisRampBreakModes.Immediate;
```

State management

The property can be read and written in any state.

30 MCAxis.Mechanism.Units

The **MCAxis.Mechanism.Units** functionality allows you to read or adjust the units used for the values of positions, speeds, accelerations and jerk.

These units are usually defined during axis definition and can be queried in the STX program using these commands.

NOTICE



Changed positions due to a change of unit during machine operation

It is possible to change the units while the machine is in operation. However, since the positioning command only contains the values without specifying a unit, this changes the meaning of the values.

Example: Changing from millimeters to inches results in a different position in the machine for a value of `25.0` in the positioning command.

- ► Avoid changing the units during operation.
- ► Change the units with care.

30.1 Syntax for MCAxis.Mechanism.Units

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Mechanism.Units.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

30.2 Functions of MCAxis.Mechanism.Units

The functionality provides the following functions:

- SetLinearUnits()
- SetRotatoryUnits()

30.2.1 SetLinearUnits()

This function allows you to change all units for positions, speeds, accelerations and jerks for a linear movement in one function. The parameters allow you to specify a different meaning for each unit.

Parameters

Parameter	Data type	Meaning
PositionUnit	**MCLinearPositionUnits** (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for the linear position values. Possible units: – Millimeters – Inches – 1/10 millimeters – 1/100 millimeters

Parameter	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 the linear speed 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 the 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 the 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 management

The function can be read in any state.

The function can be written in any state. However, it is recommended to change units only in the **Disabled** state.

30.2.2 SetRotaryUnits()

This function allows you to change all units for positions, speeds, accelerations and jerks for a rotary movement in one function. The parameters allow you to specify a different meaning for each unit.

Parameters

Parameter	Data type	Meaning
PositionUnit	**MCRotatoryPositionUnits** (optional) – MCRotatoryPositionUnits.DEG – MCRotatoryPositionUnits.RAD – MCRotatoryPositionUnits.REV	Unit for the rotary position values. Possible units: – Degrees – Radians (2 Pi) – Revolutions

Parameter	Data type	Meaning
SpeedUnit	**MCRotatorySpeedUnits** (optional) – MCRotatorySpeedUnits.DEG_PER_SEC – MCRotatorySpeedUnits.RAD_PER_SEC – MCRotatorySpeedUnits.REV_PER_MIN	Unit for the rotary speed values. Possible units: – Degrees per second – Radians (2 Pi) per second – Revolutions per minute
AccelerationUnit	**MCRotatoryAccelerationUnits** (optional) – MCRotatoryAccelerationUnits.DEG_PER_S2 – MCRotatoryAccelerationUnits.RAD_PER_S2	Unit for the rotary acceleration values. Possible units: – Degrees per second² – Radians (2 Pi) per second²
JerkUnit	**MCRotatoryJerkUnits** (optional) – MCRotatoryJerkUnits.DEG_PER_S3 – MCRotatoryJerkUnits.RAD_PER_S3	Unit for the rotary jerk values. Possible units: – Degrees per second³ – Radians (2 Pi) per second³

Syntax

```
AX1.Mechanism.Units.SetRotatoryUnits(
    MCRotatoryPositionUnits.DEG,
    MCRotatorySpeedUnits.REV_PER_MIN,
    MCRotatoryAccelerationUnits.DEG_PER_S2,
    MCRotatoryJerkUnits.DEG_PER_S3);
```

State management

The function can be read in any state.

The function can be written in any state. However, it is recommended to change units only in the **Disabled** state.

30.3 Properties of MCAxis.Mechanism.Units

The functionality provides the following properties:

– LinearPositionUnit
– LinearSpeedUnit
– LinearAccelerationUnit
– LinearJerkUnit
– RotatoryPositionUnit
– RotatorySpeedUnit
– RotatoryAccelerationUnit
– RotatoryJerkUnit

30.3.1 LinearPositionUnit

Description

Reads or writes the unit in which the position values for an axis defined as linear are processed and displayed.

Data type	Meaning
MCLinearPositionUnits (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for the linear position values. Possible units: – Millimeters – Inches – 1/10 millimeters – 1/100 millimeters

Syntax

```
PositionUnits := AX1.Mechanism.Units.LinearPositionUnit;
AX1.Mechanism.Units.LinearPositionUnit := MCLinearPositionUnits.MM_10TH;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.2 LinearSpeedUnit

Description

Reads or writes the unit in which the speed values for an axis defined as linear 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 the linear speed values. Possible units: – Millimeters per second – Millimeters per minute – Meters per second – Meters per minute – Inches per second – Inches per minute

Syntax

```
SpeedUnits := AX1.Mechanism.Units.LinearSpeedUnit;
AX1.Mechanism.Units.LinearSpeedUnit := MCLinearSpeedUnits.M_SEC;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.3 LinearAccelerationUnit

Description

Reads or writes the unit in which the acceleration values for an axis defined as linear are processed and displayed.

Data type	Meaning
MCLinearAccelerationUnits (optional) – MCLinearAccelerationUnits.MM_PER_S2 – MCLinearAccelerationUnits.M_PER_S2 – MCLinearAccelerationUnits.INCH_PER_S2	Unit for the linear speed values. Possible units: – Millimeters per second² – Meters per second² – Inches per second²

Syntax

```
AccelerationUnits := AX1.Mechanism.Units.LinearAccelerationUnit;
AX1.Mechanism.Units.LinearAccelerationUnit :=
MCLinearAccelerationUnits.M_PER_S2;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.4 LinearJerkUnit

Description

Reads or writes the unit in which the jerk values for an axis defined as linear are processed and displayed.

Data type	Meaning
MCLinearJerkUnits (optional) – MCLinearJerkUnits.MM_PER_S3 – MCLinearJerkUnits.M_PER_S3 – MCLinearJerkUnits.INCH_PER_S3	Unit for the linear speed 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 management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.5 RotaryPositionUnit

Description

Reads or writes the unit in which the position values for an axis defined as rotary are processed and displayed.

Data type	Meaning
MCRotatoryPositionUnits (optional) – MCRotatoryPositionUnits.MM – MCRotatoryPositionUnits.INCH – MCRotatoryPositionUnits.MM_10TH – MCRotatoryPositionUnits.MM_100TH	Unit for the rotary position values. Possible units: – Millimeters – Inches – 1/10 millimeters – 1/100 millimeters

Syntax

```
PositionUnits := AX1.Mechanism.Units.RotatoryPositionUnit;
AX1.Mechanism.Units.RotatoryPositionUnit := MCRotatoryPositionUnits.RAD;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.6 RotatorySpeedUnit

Description

Reads or writes the unit in which the speed values for an axis defined as rotary are processed and displayed.

Data type	Meaning
MCRotatorySpeedUnits (optional) – MCRotatorySpeedUnits.DEG_PER_SEC – MCRotatorySpeedUnits.RAD_PER_SEC – MCLinearSpeedUnits.REV_PER_MIN	Unit for the linear speed values. Possible units: – Degrees per second – Radians (2 Pi) per second – Revolutions per minute

Syntax

```
RotationalSpeedUnits := AX1.Mechanism.Units.RotatorySpeedUnit;
AX1.Mechanism.Units.RotatorySpeedUnit := MCRotatorySpeedUnits.REV_PER_MIN;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.7 RotatoryAccelerationUnit

Description

Reads or writes the unit in which the angular acceleration values for an axis defined as rotary are processed and displayed.

Data type	Meaning
MCRotatoryAccelerationUnits (optional) – MCRotatoryAccelerationUnits.DEG_PER_S2 – MCRotatoryAccelerationUnits.RAD_PER_S2	Unit for the rotary acceleration values. Possible units: – Degrees per second² – Radians (2 Pi) per second²

Syntax

```
AccelerationUnits := AX1.Mechanism.Units.LinearAccelerationUnit;
AX1.Mechanism.Units.LinearAccelerationUnit :=
MCLinearAccelerationUnits.M_PER_S2;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

30.3.8 RotatoryJerkUnit

Description

Reads or writes the unit in which the jerk values for an axis defined as rotary are processed and displayed.

Data type	Meaning
MCRotatoryJerkUnits (optional) – MCRotatoryAcceleratio-nUnits.DEG_PER_S3 – MCRotatoryAcceleratio-nUnits.RAD_PER_S3	Unit for the rotary acceleration values. Possible units: – Degrees per second³ – Radians (2 Pi) per second³

Syntax

```
JerkUnits := AX1.Mechanism.Units.RotatoryJerkUnit;
AX1.Mechanism.Units.RotatoryJerkUnit := MCRotatoryJerkUnits.MM_PER_S3;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

31 MCAxis.MoveHalt

The **MCAxis.MoveHalt** functionality allows you to stop a moving MCX axis with a ramp. Unlike the `Stop()` function, the preliminary buffers are not cleared. In buffered mode, this function is also written to the buffer and processed there.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

31.1 Syntax for MCAxis.MoveHalt

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.MoveHalt.Start.(optional <parameter 1, parameter 2>);
```

31.2 Functions of MCAxis.MoveHalt

The functionality provides the following functions:

– Start()

31.2.1 Start()

The function executes the stop command immediately. The parameters passed define how the axis is to stop.

Observe the following rules:

– The `Deceleration` parameter is entered in the user units predefined for the axis.

– Adjusting the maximum deceleration via this parameter changes the value in the axis object. If, for example, no value for the `Deceleration` parameter is passed with the next `MovePtp.Start` command, the value specified here is used.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameters

Parameter	Data type	Meaning
MCAxisHaltMode	Enum MCAxisHaltModes (optional)	Behavior of the axis after it has been decelerated with a deceleration ramp:
	MCAxisHaltModes.Normal	After the stop ramp has been completed, the axis comes to a standstill in position control.
	MCAxisHaltModes.AtActualPosition	After the stop ramp has been completed, the axis moves back to the position where it was when the `MoveHalt.Start()` function was called. The return movement is executed with the parameters set for this axis object.

Parameter	Data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
AX1.MoveHalt.Start(MCAxisHaltModes.Normal,50000);
AX1.MoveHalt.Start(MCAxisHaltModes.AtActualPosition,50000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	–		Disabled
Enabled	–		Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis has been stopped (MCAxis.State [▶ 201]).

```
When Ax1.State.IsEnabled Continue; //the axis is in position control and at
standstill.
When Ax1.State.IsRunning Continue; //the setpoint generator is generating a
movement.
```

– The **ramp status** indicates which section of the movement the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 135]).

```
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 you will find an example task for a single axis that contains various Motion API functions:

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

Task Sample2 Autorun

loop
    when run Continue;

    //Switch 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
switch-on position.
    When AX1.MoveHome.IsReferenceSet Continue;

    //Start the axis in the positive direction in continuous mode.
    AX1.MoveVelocity.Start(Directions.Positive,Speed1,5000,);
    //If the user has meanwhile cleared the auxiliary flag in the setup,
stop the axis with a ramp of 500 mm/s² and disable it.
    //This event could be a jog button, for example.
    When 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** functionality allows you to start and/or parameterize homing of an MCX axis.

On the page *MotionHome sequence diagram*, the JetSym online help describes in detail the different types of homing as well as their function and sequence.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

32.1 Syntax for MCAxis.MoveHome

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.MoveHalt.Home.<function or value access> (optional <parameter 1,
parameter 2, parameter n>);
```

32.2 Functions of MCAxis.MoveHome

The functionality provides the following functions:

– Start()
– SetReference()

Functions for the reference shift when using absolute encoders

– SetAbsEncoderReference()
– ClearAbsEncoderReferenceStatus()

32.2.1 Start()

The function starts homing of an MCX axis. The axis immediately starts moving in the direction specified with `Direction`. Its behavior when reaching the hardware limit switches depends on the selected type of homing. For example, it can reverse automatically and continue moving in the other 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.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.
– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
Direction	Directions (optional)	Defines the direction in which homing is started (see Direction [▶ 157]).
SwitchConfig	ReferencingSwitchConfigs (optional)	Defines the switch configuration used for homing (see SwitchConfig [▶ 158]).
Mode	ReferencingModes (optional)	Defines the type of homing sequence (see Mode [▶ 159]).
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning (see Acceleration [▶ 160]). Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.
SpeedSwitchSearch	Double (optional)	Defines the speed at which the axis performs its positioning to search for the limit and/or reference switches (see SpeedSwitchSearch [▶ 160]).
MaxDistanceSwitchSearch	Double (optional)	Defines the maximum distance the axis may travel until it reaches the relevant switches for homing. If this distance is exceeded, homing is aborted with an error (see MaxDistanceSwitchSearch [▶ 161]).
SpeedReferenceSearch	Double (optional)	Defines the speed at which the axis searches for the reference point after the reference/limit switch has been found (see SpeedReferenceSearch [▶ 162]).
MaxDistanceReferenceSearch	Double (optional)	Defines the maximum distance the axis may travel until it reaches the reference point after the reference/limit switch has been found. If this distance is exceeded, homing is aborted with an error (see MaxDistanceReferenceSearch [▶ 162]).
ReferenceOffset	Double (optional)	Home position: Distance: Defines the number of position units by which the axis is moved after the reference has been found, before the position is set to the home position (see ReferenceOffset [▶ 163]).
NormalPosition	Double (optional)	Home position; defines the position to which the axis is set after homing has been completed successfully (see NormalPosition [▶ 164]).

Syntax

```
AX1.MoveHome.Start(Directions.Positive,ReferencingSwitchConfigs.RefAndLim-
itSwitch,ReferencingModes.ZeroPulse_2Phase,3000,500,200,20,15,125,0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	X		Running
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.2.2 SetReference()

The function sets the current position of the MCX axis to the value passed with the function.

After the control unit is restarted, this value is `0.0`. If no value is passed for the parameter, motion control uses the value passed last.

Parameter	Data type	Meaning
Position	Double (optional)	Defines the reference position.

Syntax

```
AX1.MoveHome.SetReference(10.0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.2.3 SetAbsEncoderReference()

ⓘ Info

Reference shift when using absolute encoders

This command is useful where absolute encoders are used for the reference shift.

For this purpose, refer to the information in the JetSym online help or in the information sheet *Saving and redetermining the reference point for (multi-turn) absolute encoders (stored reference shift)* .

To use the command, the storage function in the NVRAM must be activated.

Parameters

The value passed is transferred to the parameter `Home position: Position` (`NormalPosition`). If no value is passed, the value last passed to the parameter `Home position: Position` (`NormalPosition`) is used.

Unlike the "SetReference" function, the internally calculated reference offset is stored in the internal battery-backed RAM (and on a USB stick, if present). When the control unit is booted again, this calculated value is used again for the reference shift, so that the actual position of the MCX axis can be operated directly with the reference shift defined once. The reference status is then immediately "Home position set".

Parameter	Data type	Meaning
Position	Double (optional)	Writes the parameter `Home position: Position` (`NormalPosition`).

Syntax

```
AX1.MoveHome.SetAbsEncoderReference(0.0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.2.4 ClearAbsEncoderReferenceStatus()

> **Info**
>
> **Reference shift when using absolute encoders**
>
> This command is useful where absolute encoders are used for the reference shift.
>
> For this purpose, refer to the information in the JetSym online help or in the information sheet *Saving and redetermining the reference point for (multi-turn) absolute encoders (stored reference shift)* .
>
> To use the command, the storage function in the NVRAM must be activated.

This function allows you to clear the reference status of an axis. This has the following effects:

– The reference state is set to `NoReference`.

– The query `IsAbsEncoderReferenceValid` returns `false`.

– The current reference offset remains effective until the control unit is restarted.

This function can also be used in applications without absolute encoders to set the reference state to `NoReference`.

Parameters

The function has no parameters.

Syntax

```
AX1.MoveHome.ClearAbsEncoderReferenceStatus();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3 Properties of MCAxis.MoveHome

The functionality provides the following properties:

– Direction

– SwitchConfig

– Mode

– Acceleration

– SpeedSwitchSearch

– MaxDistanceSwitchSearch
– SpeedReferenceSearch
– MaxDistanceReferenceSearch
– ReferenceOffset
– NormalPosition
– ReferenceState
– IsReferenceRunStarted
– IsSwitchFound
– IsReferenceFound
– IsReferenceSet

Functions for the reference shift when using absolute encoders

– IsAbsEncoderReferenceValid
– ReferenceShift

32.3.1 Direction

Description

Data type	Meaning
Type Directions (optional)	Reads or writes the direction in which homing is started.
Directions.positive	The axis starts homing in the positive direction (counting direction of the actual position).
Directions.negative	The axis starts homing in the negative direction (counting direction of the actual position).

Syntax

```
RefDirection := AX1.MoveHome.Direction;
AX1.MoveHome.Direction := Directions.Negative;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.2 SwitchConfig

Description

Data type	Meaning
ReferencingSwitchConfigs (optional)	Reads or writes the switch configuration used for homing.
ReferencingSwitchConfigs.NoSwitch	No switch is taken into account. The reference can only be set at the zero pulse of the encoder. This setting can be used, for example, for modulo axes without 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. 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, homing is aborted with an error.
ReferencingSwitchConfigs.RefSwitch	Only the reference switch is taken into account. If a limit switch is reached before the reference switch, homing is aborted with 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 management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.3 Mode

Description

Data type	Meaning
ReferencingModes (optional)	Reads or writes the mode used for homing.
ReferencingModes.ZeroPulse_2Phase	The reference point is determined at the zero pulse of the encoder. Homing is performed in 2 phases.
Referencing-Modes.SwitchingEdge_2Phase	The reference point is determined at a switch edge (reference or limit switch). Homing is performed in 2 phases.
ReferencingModes.ZeroPulse_1Phase	The reference point is determined at the zero pulse of the encoder. Homing is performed in 1 phase.
Referencing-Modes.SwitchingEdge_1Phase	The reference point is determined at a switch edge (reference or limit switch). Homing is performed in 1 phase.

Syntax

```
RefMode := AX1.MoveHome.Mode;
AX1.MoveHome.Mode := ReferencingModes.SwitchingEdge_2Phase;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.4 Acceleration

Description

Data type	Meaning
Double (optional)	Reads or writes the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
RefAcceleration := AX1.MoveHome.Acceleration;
AX1.MoveHome.Acceleration := 10000;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.5 SpeedSwitchSearch

Description

Data type	Meaning
Double (optional)	Reads or writes the speed at which the axis performs its positioning to search for the limit and/or reference switches.

Syntax

```
RefSpeed1 := AX1.MoveHome.SpeedSwitchSearch;
AX1.MoveHome.SpeedSwitchSearch := 250;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.6 MaxDistanceSwitchSearch

Description

Data type	Meaning
Double (optional)	Reads or writes the maximum distance the axis may travel until it reaches the relevant switches for homing. If this distance is exceeded, homing is aborted with an error.

Syntax

```
MaxReferencePosition := AX1.MoveHome.MaxDistanceSwitchSearch;
AX1.MoveHome.MaxDistanceSwitchSearch := 5000.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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. Homing without switch search as well as 1-phase homing is only performed at this speed.

Syntax

```
RefSpeed2 := AX1.MoveHome.SpeedReferenceSearch;
AX1.MoveHome.SpeedReferenceSearch := 50;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.8 MaxDistanceReferenceSearch

Description

Data type	Meaning
Double (optional)	Reads or writes the maximum distance the axis may travel until it reaches the reference point after the reference/limit switch has been found. If this distance is exceeded, homing is aborted with an error.

Syntax

```
MaxReferenceSearchPos := AX1.MoveHome.MaxDistanceReferenceSearch;
AX1.MoveHome.MaxDistanceReferenceSearch := 100.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.9 ReferenceOffset

Description

Data type	Meaning
Double (optional)	Reads or writes the number of position units by which the axis is moved after the reference has been found, before the position is set to the home position.

Syntax

```
HomePositionOffset := AX1.MoveHome.ReferenceOffset;
AX1.MoveHome.ReferenceOffset := 1265.5;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.10 NormalPosition

Description

Data type	Meaning
Double (optional)	Reads or writes the position to which the axis is set after the reference has been found. At the end of successful homing, the MCX system sets the setpoint and actual position of the axis to this value.

Syntax

```
HomePositionValue := AX1.MoveHome.NormalPosition;
AX1.MoveHome.NormalPosition := 0.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.11 ReferenceState

Description

Data type	Meaning
Type MCReferenceStates (optional)	Reads the status of homing. This information allows you to recognize whether the axis has already been homed or which part of homing the axis is in. There are 5 homing statuses that can be queried. Depending on the type of homing, some of these statuses may be skipped or not assumed.
MCReferenceStates.NoReference	– The axis has not yet been homed, i.e. no successful homing has taken place yet or the `SetReference()` function has not yet been executed. – An error occurred on an axis with the reference status `ReferenceSet` during a renewed attempt to determine the reference. – The ClearAbsEncoderReferenceStatus() [▶ 156] function has been executed.

Data type	Meaning
MCReferenceStates.Refer-enceRunStarted	Homing has been started. 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 found (transition from phase 1 to phase 2).
MCReferenceStates.Reference-Found	The reference mark has been found.
MCReferenceStates.Reference-Set	The home position has been set. The axis is thus homed and homing has finished.

Syntax

```
When AX1.MoveHome.ReferenceState == MCReferenceStates.ReferenceRunStarted
Continue;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

32.3.12 IsReferenceRunStarted, IsSwitchFound, IsReferenceFound, IsReferenceSet

Description

Data type	Meaning
Bool	The properties read whether the axis is in the queried reference status (see ReferenceState [▶ 164]). If none of these properties is fulfilled, the reference status of the axis is `NoReference`.

Syntax

```
HomingStart := AX1.MCAxis.MoveHome.IsReferenceRunStarted
```

State management

The property can be queried in any state.

32.3.13 IsAbsEncoderReferenceValid

Description

ⓘ Info

Reference shift when using absolute encoders

This command is useful where absolute encoders are used for the reference shift.

For this purpose, refer to the information in the JetSym online help or in the information sheet *Saving and redetermining the reference point for (multi-turn) absolute encoders (stored reference shift)* .

To use the command, the storage function in the NVRAM must be activated.

Data type	Meaning
Bool	Queries whether the reference shift function is active for this MCX axis. If `True`, a reference shift exists (it can also be 0) and is applied when the control unit boots.

Syntax

```
If AX1.MotionHome.IsAbsEncoderReferenceValid then …
```

State management

The property can be queried in any state.

32.3.14 ReferenceShift

Description

ⓘ Info

Reference shift when using absolute encoders

This command is useful where absolute encoders are used for the reference shift.

For this purpose, refer to the information in the JetSym online help or in the information sheet *Saving and redetermining the reference point for (multi-turn) absolute encoders (stored reference shift)* .

To use the command, the storage function in the NVRAM must be activated.

Data type	Meaning
Double (optional)	Reads or writes the number of position units by which the zero position of the MCX axis with absolute encoder is shifted relative to the zero point of the encoder. This value can also be written. **NOTICE! However, this only makes sense for very small corrections**. Preferably use the `SetAbsEncoderReference()` function to define the position at which the MCX axis is to stand.

Syntax

```
ReferenceShift := AX1.MoveHome.ReferenceShift;
AX1.MoveHome.ReferenceShift := 257.5;
```

State management

The property can be queried in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33 MCAxis.MovePtp

The **MCAxis.MovePtp** functionality allows you to start and parameterize point-to-point positioning of an MCX axis.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

33.1 Syntax for MCAxis.MovePtp

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.MovePtp.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

33.2 Functions of MCAxis.MovePtp

The functionality provides the following functions:

- Start()
- StartNewTargetPosition()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()

33.2.1 Start()

The function starts point-to-point positioning of an MCX axis. The axis starts moving immediately or applies the data to a running positioning.

Since all parameters required for positioning can be passed in a single command, this is the most compact way to fully parameterize and start a positioning movement.

Parameters

Observe the following rules:

- All parameters are specified in the units that the user has predefined for the axis.
- The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Positioning in buffered mode

In buffered mode, you can call several positionings that are processed one after the other. This means that the motion control unit starts a new positioning as soon as the setpoint of a running positioning has reached its target. Further information can be found in section MCAxis.Buffer [▶ 31].

Positioning in non-buffered mode (default)

Depending on whether the axis is moving or not, the behavior is as follows:

- Calling the function in the **Enabled** state: The movement is started.
- Calling the function in the **Running** state: The motion parameters are applied to the current OR next movement according to the following table.

Parameter	Data type	Meaning	Behavior when the function is called in the Running state
Positioning-Mode	AxisPositioning-Modes (optional)	Defines the positioning mode [▶ 176].	The mode takes effect immediately.
TargetPosition	Double (optional)	Defines the target position at which the axis is to stand after positioning has been carried out.	– If the new target position can be reached taking into account the maximum values for speed, acceleration, deceleration and jerk, it is approached directly using the new motion parameters. – Otherwise, the running positioning is completed and the newly programmed one follows directly.
Speed	Double (optional)	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration). The value must be greater than `0`.	If a new positioning is started 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 changed with the parameters of the acceleration ramp.
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	The speed ramps are executed according to the new values for acceleration and deceleration. – If the movement is currently in such a ramp, transitional forms occur that do not correspond to the set ramp type. – The transitions are always executed under monitoring of the jerk limit.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change of the target window is applied to the running positioning immediately.

Positioning phases

A positioning movement consists of the following phases:

1. Acceleration ramp

2. Constant speed

   This phase only occurs if the speed `Speed` can be reached with the specified MovePtp parameters. Otherwise, the acceleration ramp changes directly into the deceleration ramp (triangular mode).

3. Deceleration ramp

Ramp shapes

The ramp shapes of the acceleration and deceleration phase are specified by the axis parameter `MCAxis.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (MCAxis.Mechanism.Slope [▶ 135]). There are two possible settings:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration,
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = sin²-shaped acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
AX1.MovePtp.Start(AxisPositioningModes.AbsNormal,20000,500,1000,1000,0.5);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis has been stopped (MCAxis.State [▶ 201]).

```
When Ax1.State.IsEnabled Continue; //the axis is in position control and at
standstill.
When Ax1.State.IsRunning Continue; //the setpoint generator is generating a
movement.
```

– The **ramp status** indicates which section of the movement the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 135]).

```
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 query whether a defined position shortly before the target has been reached, the **target window** is used (MCAxis.IsInTargetWindow [▶ 112]).

```
When Ax1.IsInTargetWindow Continue;
```

33.2.2 StartNewTargetPosition()

The function starts point-to-point positioning of an MCX axis. The axis starts moving immediately or applies the data for changing a running positioning.

The function is a simplified variant of the `Start()` function, with which only the relevant position parameters are passed. Speed values and ramp settings for the axis are adopted.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Positioning in buffered mode

In buffered mode, you can call several positionings that are processed one after the other. This means that the motion control unit starts a new positioning as soon as the setpoint of a running positioning has reached its target. Further information can be found in section MCAxis.Buffer [▶ 31].

Positioning in non-buffered mode (default)

Depending on whether the axis is moving or not, the behavior is as follows:

– Calling the function in the **Enabled** state: The movement is started.

– Calling the function in the **Running** state: The motion parameters are applied to the current OR next movement according to the following table.

Parameter	Data type	Meaning	Behavior when the function is called in the Running status
Positioning-Mode	AxisPositioning-Modes (optional)	Defines the positioning mode [▶ 176].	The mode takes effect immediately.
TargetPosition	Double (optional)	Defines the target position at which the axis is to stand after positioning has been carried out.	– If the new target position can be reached taking into account the maximum values for speed, acceleration, deceleration and jerk, it is approached directly using the new motion parameters. – Otherwise, the running positioning is completed and the newly programmed one follows directly.
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change of the target window is applied to the running positioning immediately.

Positioning phases

A positioning movement consists of the following phases:

1. Acceleration
2. Constant speed;

   This phase only occurs if the speed `Speed` can be reached with the specified MovePtp parameters.
3. Deceleration

Ramp shapes

The ramp shapes of the acceleration and deceleration phase are specified by the axis parameter `MCAxis.Mechanism.Slope.SlopeType (Type AxisSlopeTypes)`. There are two possible settings:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration,
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = sin²-shaped acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
AX1.MovePtp.StartNewTargetPosition(AxisPositioningModes.AbsNor-
mal,20000,0.5);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis has been stopped (MCAxis.State [▶ 201]).

```
When Ax1.State.IsEnabled Continue; //the axis is in position control and at standstill.
When Ax1.State.IsRunning Continue; //the setpoint generator is generating a movement.
```

– The **ramp status** indicates which section of the movement the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 135]).

```
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 query whether a defined position shortly before the target has been reached, the **target window** is used (MCAxis.IsInTargetWindow [▶ 112]).

```
When Ax1.IsInTargetWindow Continue;
```

33.2.3 NewSpeed()

The function passes a new value for the speed of a point-to-point positioning of an MCX axis. The following applies:

– The value must be > `0`.

– If the axis is already moving (**running** state), the new speed value is applied immediately.

  For a moving axis, the transition from the current to the new speed takes place with the acceleration set for the acceleration ramp. This is also the case if the speed decreases.

– If the axis is at standstill (**enabled** state), the value is only applied with the next PtP positioning.

– The ramp type cannot be changed during positioning.

– If the ramp type *sin²* is set, the jerk limit is active for all transitions.

ⓘ Info

Bringing the axis to a standstill

To bring an axis to a standstill, only the `MoveHalt` or `Stop` commands may be used. It is **not** possible to stop an axis by passing a speed value of `0`.

Parameters

Parameter	Data type	Meaning	Behavior when the function is called in the running state
Speed	Double	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration).	The new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration and jerk.

Syntax

```
AX1.MovePtp.NewSpeed(2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33.2.4 NewAcceleration()

The function passes a new value for the maximum acceleration of a point-to-point positioning of an MCX axis.

The following applies:

– An active acceleration ramp is not changed.

– The new value is applied with the next PtP positioning.

Parameters

Parameter	Data type	Meaning
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
AX1.MovePtp.NewAcceleration(2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33.2.5 NewDeceleration()

The function passes a new value for the maximum deceleration of a point-to-point positioning of an MCX axis.

The following applies:

– The value is also used during a running PtP positioning if the target can still be reached with the new parameter.

– Otherwise, the value is applied with the next PtP positioning.

Parameters

Parameter	Data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
AX1.MovePtp.NewDeceleration(2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended

Initial state	Allowed	Final state
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33.3 Properties of MCAxis.MovePtp

The functionality provides the following properties:

– PositioningMode
– TargetPosition
– TargetWindow

33.3.1 PositioningMode

Description

This property allows you to read the currently valid positioning mode of an MCX axis or to set a new value for this axis.

The following applies:

– A new value takes effect with the next point-to-point positioning, even if the change is passed during a running positioning.
– After switching on, the last target position is calculated from the actual position at the time of switching on.

Data type	Meaning
AxisPositioningMode	Mode for the type of PtP movement.
AxisPositioningMode.AbsNormal	Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the last and the new target position.
AxisPositioningMode.RelTarget	Moves the axis further by the distance passed in the `TargetPosition` parameter. The target position is calculated from the last target position plus the value in `TargetPosition`. The travel direction results from the sign of the `TargetPosition` parameter.
AxisPositioningMode.RelActual	Moves the axis further by the distance passed in the `TargetPosition` parameter. The target position is calculated from the actual position at the start of positioning plus the value in `TargetPosition`. The travel direction results from the sign of the `TargetPosition` parameter.

Data type	Meaning
AxisPositioningMode.AbsModuloPos	Only valid for modulo axes. Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning is always **positive**. The modulo wrap-around can be passed if the new target position is smaller than the last target position.
AxisPositioningMode.AbsModuloNeg	Only valid for modulo axes. Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning is always **negative**. The modulo wrap-around can be passed if the new target position is larger than the last target position.
AxisPositioningMode.AbsModuloAuto	Only valid for modulo axes. – Positioning is started with the axis stopped: Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position **by the shortest path**. The travel direction of the positioning results from the shortest path to the target. – Positioning is started with the axis moving: Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the travel direction of the running positioning.
AxisPositioningMode.AbsModuloKeepDirection	Only valid for modulo axes. Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the travel direction of the last positioning.

Syntax

```
AX_PositioningMode := AX1.MovePtp.PositioningMode;
AX1.MovePtp.PositioningMode := AxisPositioningModes.RelTarget;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33.3.2 TargetPosition

Description

This property allows you to read the currently valid value of `TargetPosition` of an MCX axis or to write a new value for this axis. It takes effect at the start of the next point-to-point positioning.

Data type	Meaning
Double	Property for determining the target position depending on the `PositioningMode` property.

Syntax

```
Target := AX1.MovePtp.TargetPosition;
AX1.MovePtp.TargetPosition := 12000;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33.3.3 TargetWindow

Description

This property allows you to read the currently valid value of the target window setting of an MCX axis or to write a new value for this axis. It takes effect at the start of the next point-to-point positioning.

Data type	Meaning
Double	Defines when the `IsInTargetWindow` status becomes active in the area of the target position of a positioning.

Syntax

```
Targetwindow := AX1.MovePtp.TargetWindow;
AX1.MovePtp.TargetWindow := 0.7;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

33.4 Example task MCAxis.MovePtp

Here you will find an example task for a single axis that contains various Motion API functions:

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

Task Sample2 Autorun

loop
    when run Continue;

    //Switch 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
switch-on position
    When AX1.MoveHome.IsReferenceSet Continue;

    //Position the axis absolutely to position 10000 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 arrived, wait 1 second.
    Delay(t#1s);

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

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

 //If the user has meanwhile cleared the auxiliary flag in the setup,
disable the axis and wait for it to be set again.
    //Otherwise, repeat reversing the axis.
    If Not run Then
        AX1.Power.Disable();
        When AX1.State.IsDisabled Continue;
    End_if;

end_loop;

End_Task;
```

34 MCAxis.MoveVelocity

The **MCAxis.MoveVelocity** functionality allows you to start and parameterize continuous positioning of an MCX axis.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

34.1 Syntax for MCAxis.MoveVelocity

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.MoveVelocity.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

34.2 Functions of MCAxis.MoveVelocity

The functionality provides the following functions:

- Start()
- Reverse()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()

34.2.1 Start()

The function starts continuous positioning of an MCX axis. The axis starts moving immediately or applies the data to a running positioning.

The axis is accelerated with the current or programmed ramp and moves at the current or programmed value for `Speed` until it is stopped by a halt or stop command. Continuous positioning is position-controlled, so that a position following error is also monitored during the movement.

Since all parameters required for positioning can be passed in a single command, this is the most compact way to fully parameterize and start a positioning movement.

Parameters

Observe the following rules:

- All parameters are specified in the units that the user has predefined for the axis.
- The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Positioning in buffered mode

In buffered mode, you can call several positionings that are processed one after the other. This means that the motion control unit starts a new positioning as soon as the setpoint of a running positioning has reached its target. Further information can be found in section MCAxis.Buffer [▶ 31].

Positioning in non-buffered mode (default)

Depending on whether the axis is moving or not, the behavior is as follows:

– Calling the function in the **Enabled** state: The movement is started.

– Calling the function in the **Running** state: The motion parameters are applied to the current OR next movement according to the following table.

Parameter	Data type	Meaning	Behavior when the function is called in the Running state
Direction	Directions (optional)	Defines the direction of movement (see Direction [▶ 188]).	The new direction parameters are applied directly. If the new direction is opposite to the direction of the current movement, a reversing operation is executed with the acceleration ramps.
Speed	Double (optional)	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration). The value must be greater than `0`.	If a new positioning is started 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 changed with the parameters of the acceleration ramp.
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	The speed ramps are executed according to the new values for acceleration and deceleration. – If the movement is currently in such a ramp, transitional forms occur that do not correspond to the set ramp type. – The transitions are always executed under monitoring of the jerk limit.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning with the next `MoveHalt.Stop()` command if no other value is passed there. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	

Ramp shapes

The ramp shapes of the acceleration and deceleration phase are specified by the axis parameter `MCAxis.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (see MCAxis.Mechanism.Slope [▶ 135]). There are two possible settings:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration,
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = sin²-shaped acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
AX1.MoveVelocity.Start(Directions.Positive,5000,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Running
Running	√	Running	Running
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis has been stopped (MCAxis.State [▶ 201]).

```
When Ax1.State.IsEnabled Continue; //the axis is in position control and at standstill.
When Ax1.State.IsRunning Continue; //the setpoint generator is generating a movement.
```

– The **ramp status** indicates which section of the movement the setpoint generator is currently in (MCAxis.Mechanism.Slope [▶ 135]).

```
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()

The function reverses the direction of a running continuous movement and moves the axis in the new direction at the last set speed. The last set acceleration ramp is used for reversing the direction of movement.

ⓘ Info

Applicability of the `Reverse()` function

– The command cannot be applied to a running point-to-point movement (`MovePtp`).

– If the command is issued while the axis is in the **enabled** state, no action takes place.

Parameters

The function has no parameters.

Syntax

```
AX1.MoveVelocity.Reverse();
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	–		Enabled
Running	√	Running	Running
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

34.2.3 NewSpeed()

The function passes a new value for the speed of a continuous positioning of an MCX axis. The following applies:

– The value must be > `0`.

– If the axis is already moving (**running** state), the new speed value is applied immediately.

For a moving axis, the transition from the current to the new speed takes place with the values set for the acceleration ramps. This is also the case if the speed decreases.

– If the axis is at standstill (**enabled** state), the value is only applied with the next positioning.
– The ramp type cannot be changed during positioning.
– If the ramp type *sin²* is set, the jerk limit is active for all transitions.

ⓘ Info

Bringing the axis to a standstill

To bring an axis to a standstill, only the `MoveHalt` or `Stop` commands may be used. It is **not** possible to stop an axis by passing a speed value of `0`.

Parameters

Parameter	Data type	Meaning	Behavior when the function is called in the Running state
Speed	Double	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration).	If a new positioning is started 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 changed with the parameters of the acceleration ramp.

Syntax

```
AX1.MoveVelocity.NewSpeed(2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.
– the state in which calling the function is not recommended but does not trigger an error.
X the state in which calling the function is not allowed and triggers an error.

34.2.4 NewAcceleration()

The function passes a new value for the maximum acceleration of a continuous positioning of an MCX axis.

The following applies:

– An active acceleration ramp is not changed.
– The new value is applied with the next continuous positioning.

Parameters

Parameter	Data type	Meaning
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
AX1.MoveVelocity.NewAcceleration(2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

34.2.5 NewDeceleration()

The function passes a new value for the maximum deceleration of a continuous positioning of an MCX axis.

The value is used with the next `MoveHalt.Start()` command or the next positioning.

Parameters

Parameter	Data type	Meaning
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
AX1.MoveVelocity.NewDeceleration(2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

34.3 Properties of MCAxis.MoveVelocity

The functionality provides the following properties:

– Direction

34.3.1 Direction

Description

The `Direction` property allows you to read the currently valid direction of an MCX axis or to set a new value for this axis. It takes effect with the next continuous positioning. This is also the case if the change is passed during a running positioning.

Data type	Meaning
Directions (optional)	Direction of the continuous movement.
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 management

The property can be queried in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	√	Error: enabled

Initial state	Allowed	Final state
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

34.4 Example task MCAxis.MoveVelocity

Here you will find an example task for a single axis that contains various Motion API functions:

```
Var
    run: bool; //Auxiliary flag for stepping on in the program
    Direction_positive: bool;
End_Var;

Task Sample4 Autorun

Direction_positive := true;

loop
    //The flag "run" serves as a jog button. As long as it is true, the axis
moves
When run Continue;

    // Switch on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled Continue;

    //The flag "Direction_positiv" specifies 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 axis is to be stopped (run = false), call MoveHalt
    When not run Continue;
        AX1.MoveHalt.Start(MCAxisHaltModes.Normal,);
    //When the axis is at standstill, switch 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** functionality allows you to access position information of an MCX axis.

35.1 Syntax for MCAxis.Position

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Position.<value access>;
```

35.2 Functions of MCAxis.Position

The functionality provides the following functions:

Functions for MCX axes with absolute encoder

– AbsEncOverflowHandler.Activate()
– AbsEncOverflowHandler.Deactivate()

35.2.1 AbsEncOverflowHandler.Activate()

ⓘ Info

MCX axes with absolute encoders

For MCX axes with absolute encoders, there is the overflow handling functionality, for whose use the functions and properties of Motion API 2.x described here are intended.

In this context, observe the information in the JetSym help or the separate information sheets *Consistent operation of an axis with multi-turn absolute encoder (with non-binary gearbox) in case of encoder position overflow*.

Note that the storage function in the NVRAM must be activated to use these commands.

The `AbsEncOverflowHandler.Activate()` function activates the overflow handling of the MCX axis with absolute encoder.

Normally, this function is only activated once during commissioning of the axis. The setting is made in the motion setup of the axis on the Homing page. This command is therefore only required if the function has to be activated subsequently via the STX program.

Parameters

The function has no parameters.

Syntax

```
AX1.Position.AbsEncOverflowHandler.Activate();
```

State management

The function can be called in any state.

35.2.2 AbsEncOverflowHandler.Deactivate()

> **Info**
>
> **MCX axes with absolute encoders**
>
> For MCX axes with absolute encoders, there is the overflow handling functionality, for whose use the functions and properties of Motion API 2.x described here are intended.
>
> In this context, observe the information in the JetSym help or the separate information sheets *Consistent operation of an axis with multi-turn absolute encoder (with non-binary gearbox) in case of encoder position overflow*.
>
> Note that the storage function in the NVRAM must be activated to use these commands.

The `AbsEncOverflowHandler.Deactivate()` function deactivates the overflow handling of the MCX axis with absolute encoder.

Normally, this function does not need to be deactivated, unless it was activated by mistake. The setting is made in the motion setup of the axis on the Homing page.

Parameters

The function has no parameters.

Syntax

```
AX1.Position.AbsEncOverflowHandler.Deactivate();
```

State management

The function can be called in any state.

35.3 Properties of MCAxis.Position

The functionality provides the following properties:

General properties for MCX axes

- Target
- Actual
- Setpoint
- EncoderPosition
- ActualTrackingError
- TrackingErrorLimit
- TrailingIndicatorMax
- TrailingIndicatorMin

Properties specifically for MCX axes with absolute encoder

- AbsEncOverflowHandler.Deactivate() [▶ 191]
- AbsEncOverflowHandler.Activate() [▶ 190]
- AbsEncOverflowHandler.IsActive

35.3.1 Target

Description

Data type	Meaning
Double	Reads the target position of the MCX axis [in the user position unit]. The following applies: – If the axis is moving, this is the position at which the positioning ends. – If the axis is at standstill or in error, this is the target position of the most recently executed positioning command.

Syntax

```
TargetPosition := AX1.Position.Target;
```

State management

The property can be read in any state.

35.3.2 Actual

Description

Data type	Meaning
Double	Reads the position at which the MCX axis is currently located (actual position) [in the user position unit].

Syntax

```
ActualPosition := AX1.Position.Actual;
```

State management

The property can be read in any state.

35.3.3 Setpoint

Description

Data type	Meaning
Double	Reads the currently calculated setpoint position of the MCX axis [in the user position unit]. The setpoint position is recalculated by the MCX in every MCX cycle on the basis of the positioning data.

Syntax

```
SetpointPosition := AX1.Position.Setpoint;
```

State management

The property can be read in any state.

35.3.4 EncoderPosition

Description

Data type	Meaning
Double	Reads the position output by the position sensor of the MCX axis [in the user position unit]. This value differs from the actual position by offsets that are calculated during homing or, if applicable, in the case of modulo overflows. Output in user position units taking into account the mechanical ratio.

Syntax

```
EncoderRawValue := AX1.Position.EncoderPosition;
```

State management

The property can be read in any state.

35.3.5 AcutalTrackingError

Description

Data type	Meaning
Double	Reads the value of the current following error [in the user position unit]. The following error is calculated in the drive by subtracting the actual position from the setpoint position in every cycle of the position controller. It is a measure of how well the drive can follow the setpoint specifications of the MCX.

Syntax

```
FollowingError := AX1.Position.ActualTrackingError;
```

State management

The property can be read in any state.

35.3.6 TrailingIndicatorMax

Description

Data type	Meaning
Double	Reads the highest positive following error detected by the control since the measurement was last reset [in the user position unit]. The following error peak measurement can be used to detect slow changes in the following error for movements that are always the same (see also `AX1.Drive.Control.Position.ResetTrailingIndicators()`).

Syntax

```
MaxPosFollowingError := AX1.Position.TrailingIndicatorMax;
```

State management

The property can be read in any state.

35.3.7 TrailingIndicatorMin

Description

Data type	Meaning
Double	Reads the highest negative following error detected by the control since the measurement was last reset [in the user position unit]. The following error peak measurement can be used to detect slow changes in the following error for movements that are always the same (see also `AX1.Drive.Control.Position.ResetTrailingIndicators()`).

Syntax

```
MinNegFollowingError := AX1.Position.TrailingIndicatorMin;
```

State management

The property can be read in any state.

35.3.8 AbsEncOverflowHandler.IsActive

ⓘ Info

MCX axes with absolute encoders

For MCX axes with absolute encoders, there is the overflow handling functionality, for whose use the functions and properties of Motion API 2.x described here are intended.

In this context, observe the information in the JetSym help or the separate information sheets *Consistent operation of an axis with multi-turn absolute encoder (with non-binary gearbox) in case of encoder position overflow*.

Note that the storage function in the NVRAM must be activated to use these commands.

Parameters

Data type	Meaning
Bool	Queries whether the overflow handling function has been activated for this MCX axis.

Syntax

```
If AX1.Position.AbsEncOverflowHandler.IsActive then …
```

State management

The property can be read in any state.

36 MCAxis.Power

The **MCAxis.Power** functionality allows you to enable or disable the control and thus the torque generation of an MCX axis.

36.1 Syntax for MCAxis.Power

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Power.<function>(optional <parameter>;
```

36.2 Functions of MCAxis.Power

The functionality provides the following functions:

– Enable()
– Disable()
– Quickstop()

36.2.1 Enable()

The function activates the controllers of the servo amplifier and enables the voltage to the motor. Enabling the voltage puts the drive in a position to generate torque at the motor shaft. If the control is set correctly and the torque is sufficient, the axis is held at its current position, as the position setpoint is preset with the actual position value.

Enable requirements

A real axis can only be enabled if the servo amplifier is ready for operation. Query the state of the servo amplifier as follows:

```
If AX1.Drive.IsReadyForOperation Then
```

Parameters

The function has no parameters.

Syntax

```
AX1.Power.Enable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Enabled
Enabled	–	Enabled
Running	–	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, query in the program whether the new state has been reached before calling further commands for the axis.

```
When Ax1.State.IsEnabled Continue;
```

36.2.2 Disable()

The function disables the servo amplifier for torque generation (pulse inhibit). The motor is de-energized immediately and coasts to a stop if no brake becomes active.

Enable requirements

A real axis can only be enabled if the servo amplifier is ready for operation. Query the state of the servo amplifier as follows:

```
If AX1.Drive.IsReadyForOperation Then
```

Parameters

Parameter	Data type	Meaning
DisableModes	MCPowerDisableMode (optional)	Mode for executing the `Disable()` function
	MCPowerDisableMode.Forced (default)	The pulse inhibit always takes effect immediately. This mode is also active if no parameter is specified.
	MCPowerDisableMode.StateCheck	The pulse inhibit only takes effect if the axis is in the *Enabled* operating state. In all other states, an operating error is output.

Syntax

```
Ax1.Power.Disable();
Ax1.Power.Disable(MCPowerDisableModes.Forced);
Ax1.Power.Disable(MCPowerDisableModes.StateCheck);
```

State management

The following table applies to writing the function:

The following table applies to calling without a parameter or with the `MCPowerDisableModes.Forced` parameter:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	√	Disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

The following table applies to calling with the `MCPowerDisableModes.StateCheck` parameter:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, before calling further commands for the axis, query in the program whether the new state has been reached, as follows:

```
When Ax1.State.IsDisabled Continue;
```

If the axis was previously in the *Error: enabled* state, use the following command. The position buffers are cleared.

```
When AX1.State.IsDisabledError Continue;
```

36.2.3 QuickStop()

The function stops the axis with the quick stop deceleration and disables the servo amplifier once standstill has been reached.

Enable requirements

A real axis can only be enabled if the servo amplifier is ready for operation. Query the state of the servo amplifier as follows:

```
If AX1.Drive.IsReadyForOperation Then
```

Parameters

The function has no parameters.

Syntax

```
Ax1.Power.QuickStop();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	√	Disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, before calling further commands for the axis, query in the program whether the new state has been reached, as follows:

```
When Ax1.State.IsDisabled Continue;
```

36.3 Example task MCAxis.Power

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample1 Autorun
    //Switch on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled Continue;

    //a process with the axis can be placed here, or other tasks move the
axis
    //…

    // Disable the axis
    If AX1.State.IsRunning Then    // if the axis is still moving
        AX1.Power.QuickStop();    // stop with the quick stop ramp
    Else
        AX1.Power.Disable();    // otherwise disable immediately
    End_if;
    When AX1.State.IsDisabled Continue;

End_Task;
```

37 MCAxis.Settings

The **MCAxis.Settings** functionality allows you to adjust or save all parameters of an MCX axis together.

Do not adjust parameters while saving is in progress

Saving the parameters to a file is a process that is executed in the background, i.e. it extends over several MCX cycles. If parameter values are changed during saving, it is unpredictable whether the old or the new value is written to the file.

How parameter management works

Parameters saved in the motion setup of JetSym are stored in a text file on the PC and, on request, also in the file system of the control unit (subfolder Motion). The file name of the parameter file on the control unit is *MCAxis_X.ini*, where *X* stands for the number of the axis ID. The axis ID is displayed on the Axis parameters page in the motion setup of the MCX axis. In addition, the tooltip in the hardware tree shows the axis ID when the mouse pointer is on the axis.)

When the control unit boots, the MCX kernel reads the data of the *MCAxis_X.ini* file and uses it for the axis concerned. If such a file is present on the control unit, no further parameterization is required. If there is no corresponding file for the axis, the parameters are loaded with default values.

Normally, no adjustment to the file from the STX program is required.

In very few exceptional cases, however, it can make sense to read in the parameter file on the control unit again or to save the parameters. When saving, however, bear in mind that the parameter file saved in the JetSym project may then no longer match the parameters effective on the machine.

37.1 Syntax for MCAxis.Settings

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.Settings.<function>(optional <parameter>;
```

37.2 Functions of MCAxis.Settings

The functionality provides the following functions:

– SaveToFile()
– LoadFromFile()

37.2.1 SaveToFile()

Parameters

Parameter	Data type	Meaning
Suffix	Integer (optional)	Activates saving of the parameters of the MCX axis to a file in the subfolder Motion in the file system of the control unit. Depending on the parameter passed, 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. *Y* is equal to the `Suffix` parameter.

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 management

The function can be called in any state.

37.2.2 LoadFromFile()

Parameters

Parameter	Data type	Description
Suffix	Integer (optional)	Activates the adoption of the parameters of the MCX axis from a file in the subfolder Motion in the file system of the control unit. Depending on the parameter passed, different files are read. – Parameter `0` or no parameter: The data from the file *MCaxis_X.ini* is read and used for the axis concerned. – Parameter not equal to 0: The data from the file *MCaxis_TmpY.ini* is read and used for the axis concerned. *Y* is equal to the `Suffix` parameter.

Syntax

```
AX1.Settings.LoadFromFile();     //Reads the file MCAxis_1.ini (if AX1 has
axis ID 1)
AX1.Settings.LoadFromFile(3);     //Reads the file MCAxis_1_Tmp3.ini (if AX1
has axis ID 1)
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

38 MCAxis.State

The **MCAxis.State** functionality allows you to query various information about an MCX axis, such as operating mode, motion type, etc.

Basic and further information

In this context, also observe the following information:

– For technology groups, section MCAxis.State.Techno [▶ 213];
– For state transitions, section MCAxis.State.Transitions [▶ 220];
– For a detailed description of the structure of the MCX core, the JetSym online help (JetSym Motion and MotionControl > JetSym MotionControl > MotionControl basics > Operating concept > Overview: Operating concept)
– For basic information, Motion Control Application Note No. 54 (MCX motion control basics).

Operating mode

The operating mode (`OperatingMode`) indicates how the MCX axis basically works. There are five operating modes:

Operating mode	Meaning
`Inactive` (inactive)	The axis cannot accept any commands, e.g. because it is used by a group.
`Normal` (normal)	The axis can be used. Travel commands are executed in superseding mode (see MCAxis.Buffer [▶ 31]).
`Buffered` (buffered)	The axis can be used. Travel commands are executed in buffered mode (see MCAxis.Buffer [▶ 31]).
`External` (external)	The axis is an external axis. Only the current position can be read. It can only be used as a master axis in a technology group.
`Error` (error)	An operating error is pending. The operating state is retained, but no new functions and parameters can be passed.

Tab. 2: Operating modes of MCX axes

Operating state

At any time, an MCX axis is in one, and only one, clearly defined operating state (`OperatingState`). If, for example, the axis is in the **Running** operating state, the **Enabled** operating state is not active, although the axis of course has pulse enable.

In a particular state, always only a subset of all function calls is permitted. The state model for the axis is as follows:

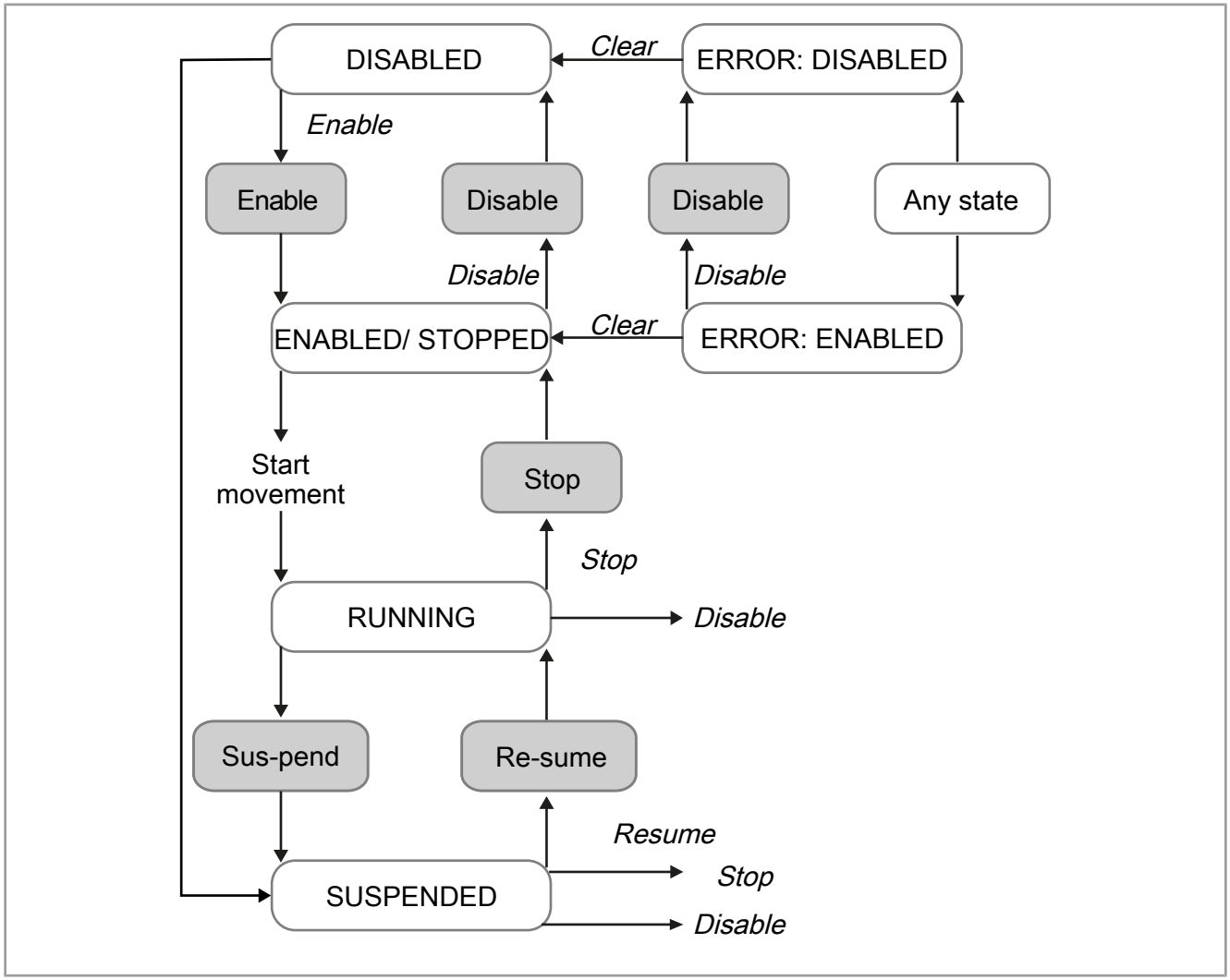

Fig. 3: State model for MCX axes

Operating state	Meaning
Inactive (inactive)	The axis cannot accept any commands, for example because it is used by a group.
Disabled (disabled)	– The controllers of the servo amplifier are open (no pulse enable). – No torque is generated.
Enabled (enabled)	– The controllers are closed (pulse enable). – The setpoint interpolator is at standstill (the preliminary buffer is empty).
Running (running)	The axis is moving, i.e. the setpoint interpolator passes motion setpoints on to the servo amplifier.
Suspended (suspended)	– Corresponds to the **Enabled** operating state, with the exception that all preliminary buffers remain filled but are locked. A movement suspended with `MCAxis.State.Transitions.Suspend()` can be continued with the command `MCAxis.State.Transitions.Resume()` (state transition resume). – In buffered mode, it is also possible to suspend (`MCAxis.Buffer.Suspend()`) and resume (`MCAxis.Buffer.Resume()`) buffer processing.
EnabledError (Error: enabled)	An error has occurred. After the error response, the axis remains in control (error level 4).

Operating state	Meaning
DisabledError (Error: disabled)	An error has occurred. After the error response, the axis disables the control (error levels 5 and 6).

Tab. 3: Operating states of MCX axes

Motion type

The motion type (`TypeOfMotion`) indicates which type of movement the setpoint generator is currently generating for the MCX axis.

Querying the motion type is useful, for example, in a monitoring task in which the process is not known. Based on the motion type active before switching off, it is possible to control the restart correctly.

Operating state	Meaning
None (`MoveNone`)	The setpoint generator outputs standstill because no movement is requested.
Positioning (`MovePtp`)	A positioning (with a programmed target) is executed.
Continuous movement (`MoveVelocity`)	A continuous positioning is executed: start with acceleration ramp, travel at programmed speed.
Homing (`MoveHome`)	Homing is executed.
Coupling active (`MoveCoupling`) [1)]	The axis is a follower axis in a technology group. The coupling process is executed.
Coupled (`MoveCoupled`) [1)]	The axis is a follower axis in a technology group. It follows the master axis according to the programmed coupling movement (electronic gearing, cam)
Decoupling active (`MoveDecoupling`) [1)]	The axis is a follower axis in a technology group. The decoupling process is executed.
Decoupled (`MoveDecoupled`) [1)]	The axis is a follower axis in a technology group. It is not connected to the master axis.
Positioning, superimposed (`MovePtpSuperPose`) [1)]	The axis is a follower axis in a technology group. It is in a superimposed positioning (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).
Continuous movement, superimposed (`MoveVelocitySuperPose`) [1)]	The axis is a follower axis in a technology group. It is in a superimposed continuous movement (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).

[1)] It is recommended to perform these queries via the technology group in which the MCX axis is active as a follower axis.

38.1 Syntax for MCAxis.State

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.State.<value access>
```

38.2 Properties of MCAxis.State

The functionality provides the following properties:

– Value
– IsDisabled
– IsEnabled
– IsRunning
– IsSupended
– IsDisabledError
– IsEnabledError
– IsInactive
– OperatingMode
– TypeOfMotion

38.2.1 Value

Description

Data type	Meaning
OperatingStates	Reads the operating state of the axis (for a detailed description, see section *Operating state* in MCAxis.State [▶ 201]).
OperatingStates.Inactive	Inactive
OperatingStates.Disabled	Disabled
OperatingStates.Enabled	Enabled
OperatingStates.Running	Running
OperatingStates.Suspended	Suspended
OperatingStates.EnabledError	Error: enabled
OperatingStates.DisabledError	Error: disabled

Syntax

```
OpState := AX1.State.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.2 IsDisabled

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Disabled** operating state.

Syntax

```
When AX1.State.IsDisabled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.3 IsEnabled

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Enabled** operating state.

Syntax

```
When AX1.State.IsEnabled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.4 IsRunning

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Running** operating state.

Syntax

```
When AX1.State.IsRunning Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.5 IsSuspended

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Suspended** operating state.

Syntax

```
When AX1.State.IsSuspended Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.6 IsDisabledError

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Error: disabled** operating state.

Syntax

```
When AX1.State.IsDisabledError Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.7 IsEnabledError

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Error: enabled** operating state.

Syntax

```
When AX1.State.IsEnabledError Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.8 IsInactive

Description

Data type	Meaning
Bool	Queries whether the axis is in the **Inactive** operating state.

Syntax

```
When AX1.State.IsInactive Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.9 OperatingMode.Value

Description

Data type	Meaning
Type MCOperatingModes	Reads the operating mode of the axis (for a detailed description, see section *Operating mode* in MCAxis.State [▶ 201]).
MCOperatingModes.Inactive	The axis is inactive, for example because it is managed by a group.

Data type	Meaning
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 mode is normally only used by axes of a path group.
MCOperatingModes.External	The axis is an external axis. It only evaluates the actual position of the second encoder of a real (internal) axis.
MCOperatingModes.Error	The axis is in an operating error.

Syntax

```
OpModeAxis := AX1.State.OperatingMode.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInInactiveMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *Inactive* operating mode.

Syntax

```
When AX1.State.OperatingMode.IsInInactiveMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInNormalMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *Normal* operating mode.

Syntax

```
When AX1.State.OperatingMode.IsInNormalMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInBufferedMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *Buffered* operating mode.

Syntax

```
When AX1.State.OperatingMode.IsInBufferedMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInExternalMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *External* operating mode.

Syntax

```
When AX1.State.OperatingMode.IsInExternalMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInErrorMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *Error* operating mode.

Syntax

```
When AX1.State.OperatingMode.IsInErrorMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.2.10 TypeOfMotion.Value

Description

Data type	Meaning
Type MCAxisTypesOfMotion	Reads the motion type of the axis (for a detailed description, see section *Motion type* in MCAxis.State [▶ 201]).
MCAxisTypesOfMotion.MoveNone	None
MCAxisTypesOfMotion.MovePtp	Positioning
MCAxisTypesOfMotion.MoveVelocity	Continuous movement
MCAxisTypesOfMotion.MoveHome	Homing
MCAxisTypesOfMotion.MoveCoupling	Coupling active
MCAxisTypesOfMotion.MoveCoupled	Coupled
MCAxisTypesOfMotion.MoveDecoupling	Decoupling active
MCAxisTypesOfMotion.MoveDecoupled	Decoupled
MCAxisTypesOfMotion.MovePtpSuperPose	Positioning (superimposed)
MCAxisTypesOfMotion.MoveVelocitySuperPose	Continuous movement (superimposed)

Syntax

```
TypeOfMotion := AX1.State. TypeOfMotion.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveNone

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `None`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveNone Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMovePtp

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Positioning`.

Syntax

```
When AX1.State.TypeOfMotion.IsMovePtp Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveVelocity

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Continuous movement`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveVelocity Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveHome

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Homing`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveHome Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveCoupling

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Coupling`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveCoupling Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveCoupled

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Coupled`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveCoupled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveDecoupling

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Decoupling`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveDecoupling Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveDeoupled

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Decoupled`.

Syntax

```
When AX1.State.TypeOfMotion.IsMoveDecoupled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMovePtpSuperPose

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Positioning (superimposed)`.

Syntax

```
When AX1.State.TypeOfMotion.IsMovePtpSuperPose Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveVelocitySuperPose

Description

Data type	Meaning
Bool	Queries whether the axis is in the motion type `Continuous movement (superimposed).`

Syntax

```
When AX1.State.TypeOfMotion.IsMoveVelocitySuperPose Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

38.3 Example task MCAxis.State

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample2 Autorun

    //Switch on the servo amplifier
    If AX1.State.IsDisabled AND AX1.Drive.IsReadyForOperation Then
        AX1.Power.Enable();
    End_if;
    When AX1.State.IsEnabled Continue;

    //A process with the axis can be placed here, or other tasks move the
axis
    //…

    // Disable the axis
    If AX1.State.IsRunning Then //if the axis is still moving
        AX1.Power.QuickStop(); //stop with the quick stop ramp
    Else
        AX1.Power.Disable(); //otherwise disable immediately
    End if;
    When AX1.State.IsDisabled Continue
End_Task;
```

39 MCAxis.State.Techno

The **MCAxis.State.Techno** functionality allows you to read the coupling status of an MCX axis that is a follower axis in a technology group (`techno`).

> **Info**
>
> **Further information**
>
> For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

39.1 Syntax for MCAxis.State.Techno

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.State.Techno.<value access>;
```

39.2 Properties of MCAxis.State.Techno

The functionality provides 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 technology group in which the MCX axis is integrated. If the technology group is not active, the value `0` is returned.

Syntax

```
ActiveTechno := AX1.State.Techno.TechnoID;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

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 involved in any active technology group, `Decoupled` is returned. If the MCX axis is the master axis in an active technology group, `Coupled` is always returned.
MCSyncStates.Decoupled	The MCX axis is decoupled.
MCSyncStates.Decoupling	The MCX axis is in the decoupling process.
MCSyncStates.Coupling	The MCX axis is in the coupling process.
MCSyncStates.Coupled	The MCX axis is coupled.

Syntax

```
If AX1.State.Techno.SyncState == MCSyncStates.Coupled then …
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.3 IsDecoupled

Description

Data type	Meaning
Bool	Reads whether the MCX axis is decoupled in an active technology group. If the MCX axis is not involved in any active technology group, `IsDecoupled` is also `true`.

Syntax

```
If AX1.State.Techno.IsDecoupled then …
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.4 IsDecoupling

Description

Data type	Meaning
Bool	Reads whether the MCX axis is in the decoupling process in an active technology group.

Syntax

```
If AX1.State.Techno.IsDecoupling then …
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.5 IsCoupling

Description

Data type	Meaning
Bool	Reads whether the MCX axis is in the coupling process in an active technology group.

Syntax

```
If AX1.State.Techno.IsCoupling then …
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.6 IsCoupled

Description

Data type	Meaning
Bool	Reads whether the MCX axis is coupled in an active technology group. If the MCX axis is the master axis in an active technology group, `IsCoupled` is also `true`.

Syntax

```
If AX1.State.Techno.IsCoupled then …
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.7 SyncPosition

Description

Data type	Meaning
Double	Reads the position at which a decoupled follower axis would have to be in order to be ready for coupling without offset. Querying this property is only of interest in the decoupled state. It allows you to recognize which follower axis position belongs to the current master axis position, in order to move the follower axis, for example, to the synchronization position where it can be coupled without offset. If the MCX axis is coupled or acts as the master axis, `SyncPosition` corresponds to the setpoint position of the follower axis.

Syntax

```
Sync_Pos := AX1.State.Techno.SyncPosition;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.8 Cam.CamID

> **Info**
>
> **Further information**
>
> For information on how a cam works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property contains several further properties that allow you to read information about an active cam in which the MCX axis is involved.

The queries only make sense for axes that are follower axes in a technology group:

– If the MCX axis is not involved in any active cam, the value `0` is returned for all read accesses to these properties.

– The value `0` is also returned for all read accesses to these properties if the MCX axis is the master axis in the technology group.

Description

Data type	Meaning
Integer	Reads the number of the active cam in which the MCX axis is involved. If the technology group concerned is not active, the value `0` is returned.

Syntax

```
ActiveCAM := AX1.State.Techno.Cam.CamID;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.9 Cam.CamSegmentID

> **Info**
>
> **Further information**
>
> For information on how a cam works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property contains several further properties that allow you to read information about an active cam in which the MCX axis is involved.

The queries only make sense for axes that are follower axes in a technology group:

– If the MCX axis is not involved in any active cam, the value `0` is returned for all read accesses to these properties.

– The value `0` is also returned for all read accesses to these properties if the MCX axis is the master axis in the technology group.

Description

Data type	Meaning
Integer	Reads the number of the active segment of the active cam in which the MCX axis is involved. If the technology group is not active, the value `0` is returned.

Syntax

```
ActiveCamSegment := AX1.State.Techno.Cam.CamSegementID;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.10 Cam.MasterOffset

ⓘ Info

Further information

For information on how a cam works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property contains several further properties that allow you to read information about an active cam in which the MCX axis is involved.

The queries only make sense for axes that are follower axes in a technology group:

– If the MCX axis is not involved in any active cam, the value `0` is returned for all read accesses to these properties.

– The value `0` is also returned for all read accesses to these properties if the MCX axis is the master axis in the technology group.

Description

Data type	Meaning
Double	Reads the offset of the master axis that was passed at the start of the cam coupling for the active cam in which the MCX axis is involved. If the technology group or the cam is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveMasterOffset := AX1.State.Techno.Cam.MasterOffset;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.11 Cam.FollowerOffset

ⓘ Info

Further information

For information on how a cam works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property contains several further properties that allow you to read information about an active cam in which the MCX axis is involved.

The queries only make sense for axes that are follower axes in a technology group:

– If the MCX axis is not involved in any active cam, the value `0` is returned for all read accesses to these properties.

– The value `0` is also returned for all read accesses to these properties if the MCX axis is the master axis in the technology group.

Description

Data type	Meaning
Double	Reads the offset of the follower axis that was passed at the start of the cam coupling for the active cam in which the MCX axis is involved. If the technology group or the cam 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 management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.12 Cam.MasterFactor

ⓘ Info

Further information

For information on how a cam works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property contains several further properties that allow you to read information about an active cam in which the MCX axis is involved.

The queries only make sense for axes that are follower axes in a technology group:

– If the MCX axis is not involved in any active cam, the value `0` is returned for all read accesses to these properties.

– The value `0` is also returned for all read accesses to these properties if the MCX axis is the master axis in the technology group.

Description

Data type	Meaning
Double	Reads the factor of the master axis that was passed at the start of the cam coupling for the active cam in which the MCX axis is involved. If the technology group or the cam is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveMasterFactor := AX1.State.Techno.Cam.MasterFactor;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

39.2.13 Cam.FollowerFactor

ⓘ Info

Further information

For information on how a cam works in a technology group, please refer to **Motion Control Application Notes (APN) Nos. 52** and **56**.

The `Cam` property contains several further properties that allow you to read information about an active cam in which the MCX axis is involved.

The queries only make sense for axes that are follower axes in a technology group:

– If the MCX axis is not involved in any active cam, the value `0` is returned for all read accesses to these properties.
– The value `0` is also returned for all read accesses to these properties if the MCX axis is the master axis in the technology group.

Description

Data type	Meaning
Double	Reads the factor of the follower axis that was passed at the start of the cam coupling for the active cam in which the MCX axis is involved. If the technology group or the cam is not active, or if the MCX axis is a master axis, the value `0` is returned.

Syntax

```
ActiveFollowerFactor := AX1.State.Techno.Cam.FollowerFactor;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

40 MCAxis.State.Transitions

The **MCAxis.State.Transitions** functionality allows you to control state transitions of an MCX axis for special applications.

As a rule, the state transitions of an MCX axis are triggered automatically by the motion commands and the behavior of the axis movement. Direct control of the state management is only required in exceptional cases, e.g. to respond in the event of an error. The **MCAxis.State.Transitions** functionality, as well as `Diagnostics.ClearErrors()`, allow operating state transitions to be triggered explicitly.

ⓘ Info

When the commands take effect

Even in buffered mode, the functions and properties of **MCAxis.State.Transitions** take effect immediately. All preliminary buffers are cleared.

The **MCAxis.State** functionality allows you to query various information about an MCX axis, such as operating mode, motion type, etc.

Basic and further information

In this context, also observe the following information:

– For technology groups, section MCAxis.State [▶ 201];
– For state transitions, section MCAxis.State.Techno [▶ 213];
– For a detailed description of the structure of the MCX core, the JetSym online help (JetSym Motion and MotionControl > JetSym MotionControl > MotionControl basics > Operating concept > Overview: Operating concept)
– For basic information, Motion Control Application Note No. 54 (MCX motion control basics).

Operating mode

The operating mode (`OperatingMode`) indicates how the MCX axis basically works. There are five operating modes:

Operating mode	Meaning
`Inactive` (inactive)	The axis cannot accept any commands, e.g. because it is used by a group.
`Normal` (normal)	The axis can be used. Travel commands are executed in superseding mode (see MCAxis.Buffer [▶ 31]).
`Buffered` (buffered)	The axis can be used. Travel commands are executed in buffered mode (see MCAxis.Buffer [▶ 31]).
`External` (external)	The axis is an external axis. Only the current position can be read. It can only be used as a master axis in a technology group.
`Error` (error)	An operating error is pending. The operating state is retained, but no new functions and parameters can be passed.

Tab. 4: Operating modes of MCX axes

Operating state

At any time, an MCX axis is in one, and only one, clearly defined operating state (`OperatingState`). If, for example, the axis is in the **Running** operating state, the **Enabled** operating state is not active, although the axis of course has pulse enable.

In a particular state, always only a subset of all function calls is permitted. The state model for the axis is as follows:

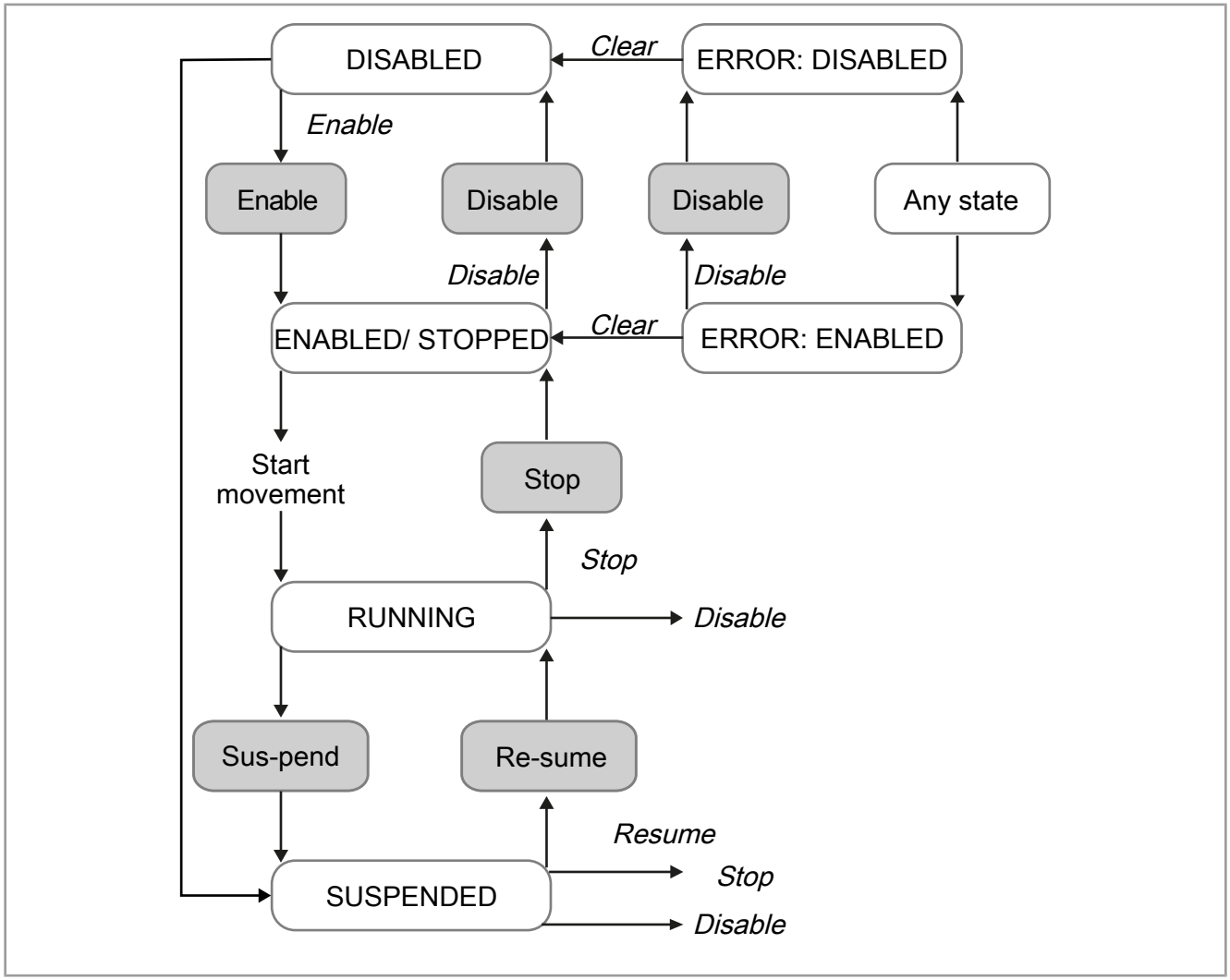

Fig. 4: State model for MCX axes

Operating state	Meaning
Inactive (inactive)	The axis cannot accept any commands, for example because it is used by a group.
Disabled (disabled)	– The controllers of the servo amplifier are open (no pulse enable). – No torque is generated.
Enabled (enabled)	– The controllers are closed (pulse enable). – The setpoint interpolator is at standstill (the preliminary buffer is empty).
Running (running)	The axis is moving, i.e. the setpoint interpolator passes motion setpoints on to the servo amplifier.
Suspended (suspended)	– Corresponds to the **Enabled** operating state, with the exception that all preliminary buffers remain filled but are locked. A movement suspended with `MCAxis.State.Transitions.Suspend()` can be continued with the command `MCAxis.State.Transitions.Resume()` (state transition resume). – In buffered mode, it is also possible to suspend (`MCAxis.Buffer.Suspend()`) and resume (`MCAxis.Buffer.Resume()`) buffer processing.

Operating state	Meaning
EnabledError (Error: enabled)	An error has occurred. After the error response, the axis remains in control (error level 4).
DisabledError (Error: disabled)	An error has occurred. After the error response, the axis disables the control (error levels 5 and 6).

Tab. 5: Operating states of MCX axes

Motion type

The motion type (`TypeOfMotion`) indicates which type of movement the setpoint generator is currently generating for the MCX axis.

Querying the motion type is useful, for example, in a monitoring task in which the process is not known. Based on the motion type active before switching off, it is possible to control the restart correctly.

Operating state	Meaning
None (`MoveNone`)	The setpoint generator outputs standstill because no movement is requested.
Positioning (`MovePtp`)	A positioning (with a programmed target) is executed.
Continuous movement (`MoveVelocity`)	A continuous positioning is executed: start with acceleration ramp, travel at programmed speed.
Homing (`MoveHome`)	Homing is executed.
Coupling active (`MoveCoupling`) [1)]	The axis is a follower axis in a technology group. The coupling process is executed.
Coupled (`MoveCoupled`) [1)]	The axis is a follower axis in a technology group. It follows the master axis according to the programmed coupling movement (electronic gearing, cam)
Decoupling active (`MoveDecoupling`) [1)]	The axis is a follower axis in a technology group. The decoupling process is executed.
Decoupled (`MoveDecoupled`) [1)]	The axis is a follower axis in a technology group. It is not connected to the master axis.
Positioning, superimposed (`MovePtpSuperPose`) [1)]	The axis is a follower axis in a technology group. It is in a superimposed positioning (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).
Continuous movement, superimposed (`MoveVelocitySuperPose`) [1)]	The axis is a follower axis in a technology group. It is in a superimposed continuous movement (see MCTechno.MovePtp.StartSuperPose() [▶ 479]).

[1)] It is recommended to perform these queries via the technology group in which the MCX axis is active as a follower axis.

40.1 Syntax for MCAxis.State.Transitions

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.State.Transitions.<function>(optional <parameter>);
```

40.2 Functions of MCAxis.State.Transitions

The functionality provides the following functions:

- Enable()
- Disable()
- ErrorDisable()
- ErrorStop()
- ErrorLockUserInput()
- Suspend()
- Resume()
- UserPositionReset()

40.2.1 Enable()

The `State.Transistions.Enable()` function is identical to the `MCAxis.Power.Enable()` function. A detailed description of the function can be found in section MCAxis.Power.Enable() [▶ 195].

Syntax

```
AX1.State.Transitions.Enable();
```

40.2.2 Disable()

Depending on the parameter passed (see the following table), the function performs the same action as the commands `MCAxis.Power.Disable()` and `MCAxis.Power.Qickstop()`. For details, refer to the descriptions of MCAxis.Power [▶ 195].

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionDisableModes	Mode for executing the `Disable()` function
	MCStateTransitionDisable-Modes.ImmediatePulseBlocking	Performs the same function as the command `MCAxis.Power.Disable()` or `MCAxis MCAxis.Power.Disable(MCPowerDisable-Modes.Forced)`
	MCStateTransitionDisable-Modes.VelocityControlSlope	Performs the same function as the command `AX1.Power.QuickStop()`.

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 that leads to the **Error: disabled** state. The error response can be set (see property ErrorDisableMode [▶ 228]). In buffered mode, the preliminary buffer is cleared.

Parameters

The function has no parameters.

Syntax

```
AX1.State.Transitions.ErrorDisable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: disabled
Enabled	√	Error: disabled
Running	√	Error: disabled
Suspended	√	Error: disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

40.2.4 ErrorStop()

This function triggers an error in the MCX axis that leads to the **Error: enabled** state. The error response can be set (see property ErrorStopMode [▶ 228]). In buffered mode, the preliminary buffer is cleared.

Parameters

The function has no parameters.

Syntax

```
AX1.State.Transitions.ErrorStop();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: enabled
Enabled	√	Error: enabled
Running	√	Error: enabled
Suspended	√	Error: enabled
Error: enabled	-	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

40.2.5 ErrorLockUserInput()

This function triggers an operating error in the MCX axis. In buffered mode, the preliminary buffer is not cleared.

Parameters

The function has no parameters.

Syntax

```
AX1.State.Transitions.ErrorLockUserInput();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

40.2.6 Suspend()

This function suspends a running positioning. The axis is stopped either with the last programmed deceleration ramp or with the configured maximum deceleration. The state after stopping is not *Enabled* but *Suspended*. An axis in this state can be moved to the original target position with the `Resume()` function.

If the MCX axis can no longer be stopped before the programmed target position with the specified deceleration, the command is ignored and the positioning is completed. The state is then *Enabled* and not *Suspended*. In buffered mode, the preliminary buffer is not cleared.

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionStopModes	Mode for the deceleration ramp of the `Suspend()` function
	MCStateTransitionStopModes.ProgDeceleration	Uses the last programmed deceleration of the MCX axis.
	MCStateTransitionStopModes.MaxDeceleration	Uses the maximum deceleration configured for the MCX axis.

Syntax

```
AX1.State.Transitions.Suspend(MCStateTransitionStopModes.ProgDeceleration);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	X	Enabled
Running	√	Suspended
Suspended	–	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

40.2.7 Resume()

The function continues a positioning that was suspended with the Suspend() [▶ 225] function. The originally specified target position is approached. The speed and acceleration values are also adopted from the original positioning.

The `Resume()` function only takes effect if the MCX axis is in the *Suspended* state.

In buffered mode, the preliminary buffer is not cleared.

Parameters

The function has no parameters.

Syntax

```
AX1.State.Transitions.Resume();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	X	Enabled
Running	–	Running
Suspended	√	Running
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

40.2.8 UserPositionReset()

The function performs a special function for resetting the setpoint generator. **Calling this function only makes sense when the axis is no longer moving.**

Executing the function aborts any running positioning and overwrites the setpoint of the MCX axis position with the actual value. In this way, the position following error and the speed setpoint immediately become zero, which immediately stops a moving axis.

A typical use case is homing to a mechanical stop. In this case, with a reduced torque limit, a positioning is executed to a position that is safely beyond the stop. When the axis is no longer moving, the `UserPositionReset()` function is executed to bring the setpoint generator to standstill at the stop position. The reference position is then defined with `SetReference()`.

After the MCAxis.Drive.TorqueShutOff [▶ 103] functionality has run, the `UserPositionReset()` function is also used.

Parameters

The function has no parameters.

Syntax

```
AX1.State.Transitions.Resume();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Enabled
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

40.3 Properties of MCAxis.State.Transitions

The functionality provides the following properties:

– ErrorDisableMode

– ErrorStopMode

40.3.1 ErrorDisableMode

Description

Data type	Meaning
MCStateTransitionDisableModes	Error response used when the `ErrorDisable()` function is triggered.
MCStateTransitionDisableModes.Velocity-ControlSlope	Stops the axis with the quick stop deceleration. When stand-still is reached, the servo amplifier is disabled.
MCStateTransitionDisableModes.Immedi-atePulseBlocking	Immediately disables the servo amplifier for torque generation (pulse inhibit).

Syntax

```
AX1.State.Transitions.ErrorDisableMode := MCStateTransitionDisableModes.Im-
mediatePulseBlocking;
ErrorResponse := AX1.State.Transitions.ErrorDisableMode;
```

State management

The property can be read and written in any state.

On read access to the property, the currently valid error response for the command is returned.

40.3.2 ErrorStopMode

Description

Data type	Meaning
MCStateTransitionErrorStopModes	Error response used for the `ErrorStop()` function
MCStateTransitionErrorStop-Modes.ProgDeceleration	Stops the axis with the programmed deceleration.
MCStateTransitionErrorStop-Modes.MaxDeceleration	Stops the axis with the maximum deceleration.
MCStateTransitionErrorStopModes.At-BlockEnd	In buffered mode: Stops the axis at the block end.

Syntax

```
AX1.State.Transitions.ErrorStopMode := MCStateTransitionStopModes.MaxDecel-
eration;
ErrorResponse := AX1.State.Transitions.ErrorStopMode;
```

State management

The property can be read and written in any state.

41 MCAxis.Stop

The **MCAxis.Stop** functionality allows you to stop a moving MCX axis with a defined ramp.

Unlike the `MoveHalt.Start()` function, `Stop` clears all preliminary buffers.

The functionality has no functions or properties.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

41.1 Functions of MCAxis.Stop

`MCAxis.Stop` is a function of the axis object.

41.1.1 Stop()

Parameters

Observe the following behavior:

– The function executes the stop command immediately. The parameter passed (see table) specifies the maximum deceleration for this.

– If no parameter is passed, the `StopMode` passed last is used.

– This parameter can also be changed by the commands `MCAxis.State.Transitions.Suspend(..)` or `MCAxis.State.Transitions.Stop(..)`.

Parameter	Data type	Meaning
StopModes	Enum StopModes (optional)	Two deceleration values are available.
	StopModes.ProgDeceleration	The deceleration last specified for the axis is used for stopping (default).
	StopModes.MaxDeceleration	The maximum acceleration of the axis is used as deceleration for stopping. It was either specified during commissioning in the motion setup under "Maximum deceleration" or programmed with the command AX1.Mechanism.Limitations.MaxAcceleration.

Syntax

```
AX1.Stop(StopModes.ProgDeceleration);
AX1.Stop(StopModes.MaxDeceleration);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√		Enabled

Initial state	Allowed	During execution	Final state
Running	√	Running	Enabled
Suspended	√		Enabled
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning.

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis is stopped.

```
When Ax1.State.IsEnabled Continue; //The axis is in position control and at
standstill.
When Ax1.State.IsRunning Continue; //The setpoint generator is generating a
movement.
```

– The **ramp status** indicates which section of the movement 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;
```

– The **target window status** is not affected by this function.

41.2 Example task MCAxis.Stop

Here you will find an example task for a single axis that contains various Motion API functions:

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

Task Sample2 Autorun

loop
    When run Continue;

    // Switch 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
switch-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 meanwhile cleared the auxiliary flag in the setup,
stop the axis with the programmed maximum deceleration
    //and disable it
    //This event could be, for example, the actuation of the emergency stop
switch
    When 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** functionality allows you to save the position of the MCX axis very quickly based on the signal of a sensor.

Typical use cases:

– Position corrections based on sensor signals (e.g. print mark correction)
– Position detection (e.g. irregularly distributed products on a conveyor belt)
– Length measurement of products

For this function, the servo amplifier provides one or two digital inputs, depending on the device family, which allow a particularly fast response. Since up to 2 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 2 touch probe channels

If only one channel is required on a servo amplifier with two touch probe channels, you should always use the first channel, as internal processing is faster for the first channel than for the second.

The position can be saved on the rising or the 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 again on the falling edge.

Single mode and continuous mode

In **single mode**, the function for a position capture is started from the STX program and the saved position is processed before a new capture is started. In most cases, this is the appropriate mode.

In single mode, a position window can be specified that limits the position range in 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 passed (in the positive or negative direction) and no signal is detected within the position window, the status "noEventInWindow" is assumed.

In **continuous mode**, on the other hand, the function is permanently active, and the saved position is overwritten with every sensor signal.

Restrictions

– For a normal axis, the value for the upper position window limit must always be greater than the value for the lower position window limit.
– For a modulo axis, if the value for the lower position window limit is greater than the value for the upper position window limit, the position window is placed across the zero point (modulo wrap-around).
– The position window limits must be within or identical to the limits of the travel range of the MCX axis. If the window function is not to be used, it makes sense to set the position window limits equal to the travel range limits.
– Only in *single mode*: When the *Both edges* configuration is used, the status only changes from *wait* to *both found* when both edges have been detected. No intermediate status *rising edge found* or *falling edge found* is assumed.
– In *continuous mode*, position window monitoring cannot be used.
– In *continuous mode*, the configuration for both edges (*both edges*) cannot be used.

Example flowcharts for various configurations

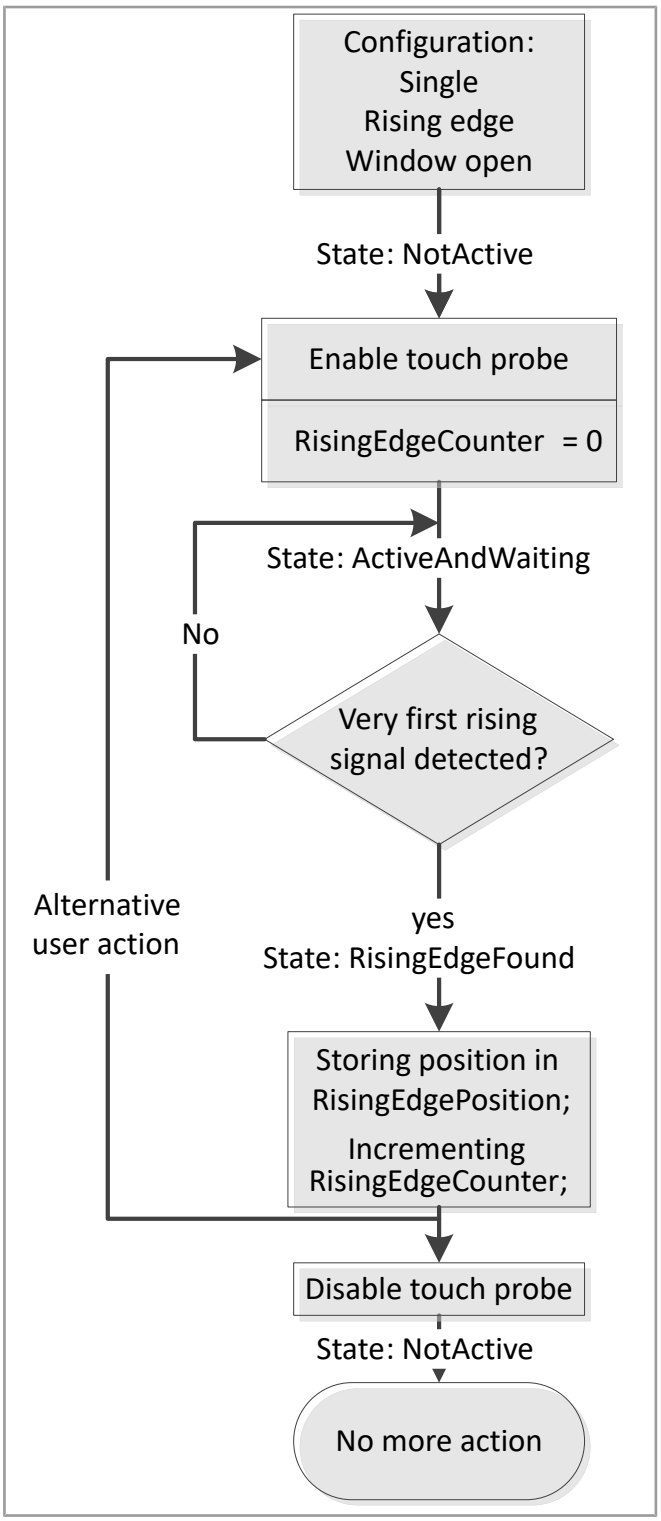

Fig. 5: *Flowchart for touch probe configuration single mode, rising edge, open position window*

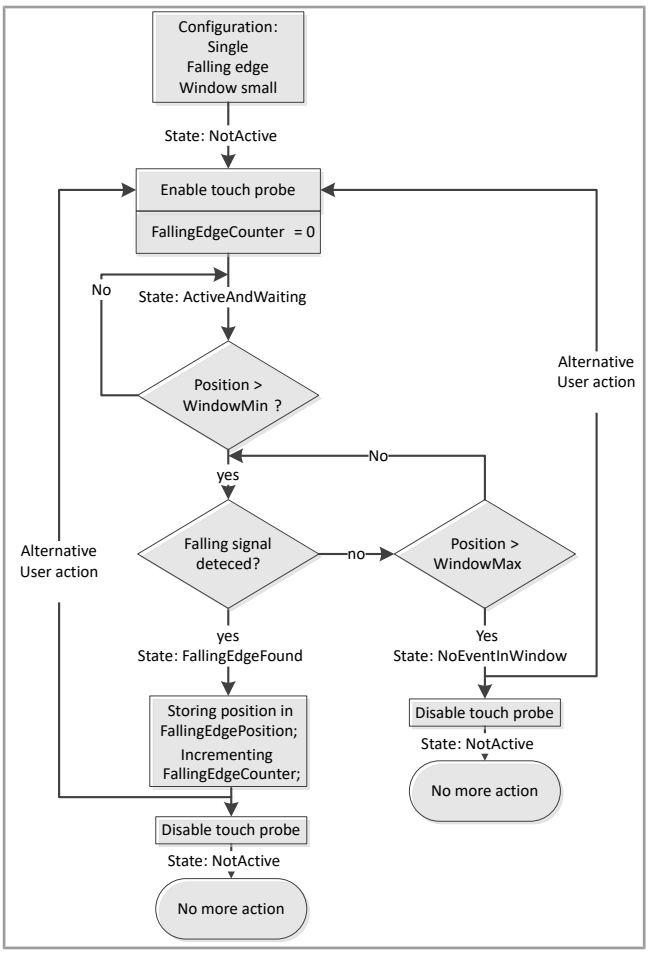

Fig. 6: *Flowchart for touch probe configuration single mode, falling edge, small position window*

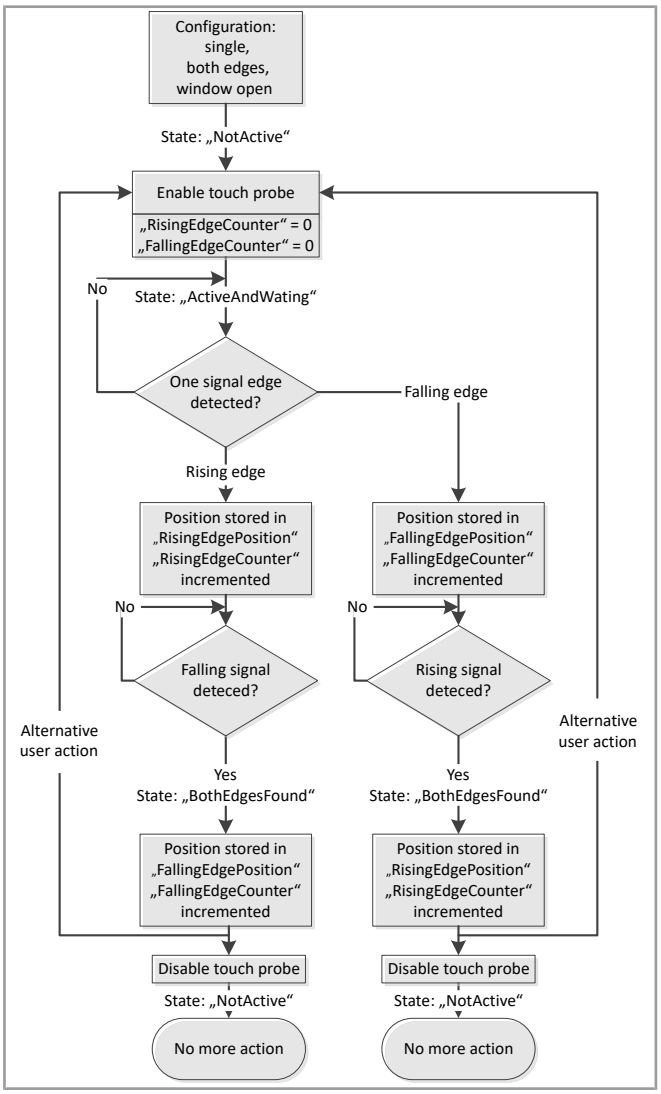

Fig. 7: *Flowchart for touch probe configuration single mode, both edges, open position window*

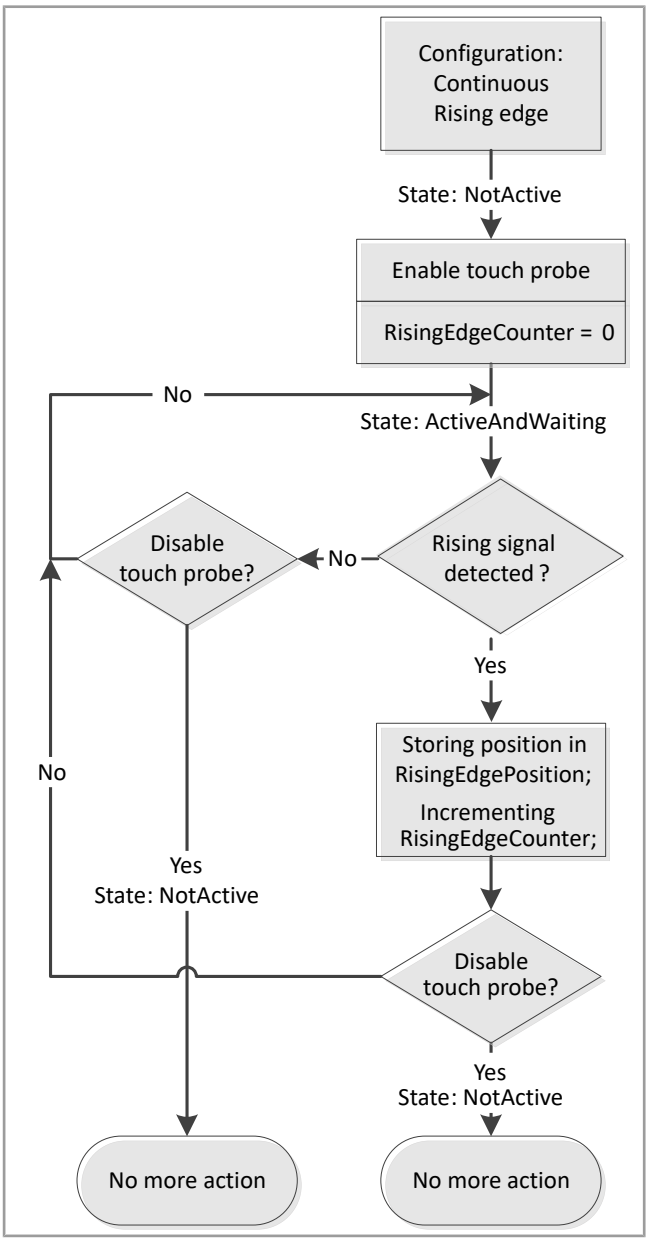

Fig. 8: *Flowchart for touch probe configuration continuous mode, rising edge*

Touch probe process

The procedure for using this function is as follows:

1. Configure the touch probe application by defining the following parameters:
   – Mode (`single`, `continuous`)
   – Edge detection (`rising`, `falling`, `both`)
   – Position window limits (`WindowMin`, `WindowMax`)
2. Activate the touch probe function with the `Enable` command.

   ⇨ The edge counter (*edge counter*) is automatically reset to zero.
3. Query the touch probe status in **single mode**.

   ⇨ The status is initially *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 with an `Enable` command. Wait at least two MCX cycles (typically 4 ms) before calling the touch probe `Enable` command again.

   OR

5. End the touch probe function permanently with a `Disable` command.
6. Query the touch probe status in **continuous mode**.
   ⇨ The status is initially *active and wait*.
   ⇨ The edge counter for the configured edge (…*EdgeCounter*) has changed when:
   – The current actual position of the MCX axis has been saved in the "..EdgePosition" parameter.
7. End the touch probe function with the `Disable` command.

⇨ The touch probe process is complete.

42.1 Syntax for MCAxis.TouchProbe

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<axis name>.TouchProbe[n].<function>(optional <parameter>);
```

42.2 Functions of MCAxis.TouchProbe

The functionality provides the following functions:

– Configure()
– Enable()
– Disable()
– ReadConfiguration()
– ReadState()

42.2.1 Configure()

This function passes the information that is suitable for your application to the **TouchProbe** functionality.

Parameters

Parameter	Data type	Meaning
Edge	MCAxisTouchProbeEdges	Signal edge to be detected.
	MCAxisTouchProbeEdges.Rising	Rising edge
	MCAxisTouchProbeEdges.Falling	Falling edge
	MCAxisTouchProbeEdges.Both	Both edges
Mode	Type MCAxisTouchProbeModes	*Touch probe mode* to be used.
	MCAxisTouchProbeModes.Trigger-FirstEvent	*Single mode*: one-time capture only.
	MCAxisTouchProbeModes.Continuous	*Continuous mode*: permanent evaluation of the edge
WindowMin	Double	Lower position window limit
WindowMax	Double	Upper position window limit

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 management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

42.2.2 Enable()

This function activates the touch probe function.

The MCX core then resets the touch probe counters (`RisingEdgeCounter`, `FallingEdgeCounter`) to zero and sets the touch probe status to *ActiveAndWaiting*.

In *single mode*, you can also use the `Enable()` function to reactivate the touch probe function without a previous `Disable()` command.

Parameters

This function has no parameters.

Syntax

```
AX1.TouchProbe[1].Enable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

42.2.3 Disable()

This function ends the touch probe function.

Parameters

This function has no parameters.

Syntax

```
AX1.TouchProbe[1].Disable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

42.2.4 ReadConfiguration()

This function allows you to read the currently active touch probe configuration. The function passes data of the structure `MCAxisTouchProbeConfiguration`. The individual elements of the structure can also be read.

Parameters

Parameter	Data type	Meaning
Edge	MCAxisTouchProbeEdges	Signal edge to be detected.
	MCAxisTouchProbeEdges.Rising	Rising edge
	MCAxisTouchProbeEdges.Falling	Falling edge
	MCAxisTouchProbeEdges.Both	Both edges

Parameter	Data type	Meaning
Mode	Type MCAxisTouchProbe-Modes	Touch probe mode to be used.
	MCAxisTouchProbeModes.Trig-gerFirstEvent	Single mode: one-time capture only.
	MCAxisTouchProbe-Modes.Continuous	Continuous mode: permanent evaluation of the edge
WindowMin	Double	Lower position window limit
WindowMax	Double	Upper position window limit

Syntax

```
TPActConfiguration := AX1.TouchProbe[1].ReadConfiguration();

TPConfiguredEdge := AX1.TouchProbe[1].ReadConfiguration().Edge;
```

State management

The following table applies to reading the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

42.2.5 ReadState()

This function allows you to read the currently active touch probe status. The function returns data of a structure of type `MCAxisTouchProbeState`. The individual elements of the structure can also be read.

Parameters

Parameter	Data type	Meaning
Triggerstate	MCAxisTouchProbeTrigger-States	Status in the sequence of 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 yet been detected in the position window.
	MCAxisTouchProbeTrigger-States.RisingEdgeFound	The touch probe function is configured for rising edge and a rising edge has been detected. The current position of the MCX axis was then copied to `RisingeEdgePosition` and `RisingEdgeCounter` was incremented
	MCAxisTouchProbeTrigger-States.FallingEdgeFound	The touch probe function is configured for falling edge and a falling edge has been detected. The current position of the MCX axis was then copied to `FallingEdgePosition` and `FallingEdgeCounter` was incremented.
	MCAxisTouchProbeTrigger-States.BothEdgesFound	The touch probe function is configured for both edges and both edges have been detected one after the other. On the signal of the rising edge, the current position of the MCX axis was copied to `RisingeEdgePosition` and `RisingEdgeCounter` was incremented. On the signal of the falling edge, the current position of the MCX axis was copied to `FallingEdgePosition` and `FallingEdgeCounter` was incremented.
	MCAxisTouchProbeTrigger-States.NoEventInWindow	The actual position has entered the position window once and left it again. However, the touch probe function has not detected the configured signal edge.
RisingEdge-Counter	Int	Number of detected rising edges of 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 has been activated, the value is undefined until a rising edge has been detected (specified in the user position unit).
FallingEdge-Counter	Int	Number of detected falling edges of the sensor signal. In single mode, only the values 0 and 1 occur.
FallingEdge-Position	Double	Copied actual position at the falling edge of the sensor signal. After the function has been activated, the value is undefined until a falling edge has been detected (specified in the user position unit).

Syntax

```
TPActStatus := AX1.TouchProbe[1].ReadState();

If AX1.TouchProbe[1].ReadState().TriggerState ==
MCAxisTouchProbeTriggerStates.RisingEdgeFound then
```

State management

The following table applies to reading the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

42.3 Example task MCAxis.TouchProbe

Here you will find an example task for a single axis that represents the sequence in the second flowchart above:

```
Var
    Run             : bool; //starts and stops the entire process (set by
user)
    TriggerdPosition: Double;
    SignalState     : Enum TypeSignalState (undefined= 0, ok, notFound);
End_Var;

Task TouchProbe Autorun

AX1.Diagnostics.ClearErrors();
AX1.MoveHome.SetReference(5.0);

//AX1: modulo axis: modulo range = 0°.. 360°
//Touchprobe 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 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** functionality (coupling axis group) allows you to define and use the position of one axis as a subordinate position processing of another axis. Possible areas of application are:

1. Moving mechanically coupled axes independently

   There are mechanical arrangements in which two axes are coupled via the mechanics in such a way that the second axis also rotates when the first is moved. To prevent the position of the second axis from changing for the application, the position of the first axis can be made subordinate to the second. In this way, the two axes are "decoupled" from the user's point of view and can be moved independently of each other.

2. (Print mark) correction of the position of an axis during movement

   By coupling a virtual axis with a real axis, you can correct the reference position of the real axis during the movement based on external information (e.g. a print mark). Since the movement of the subordinate virtual axis is executed with speed, acceleration and jerk limitation (it is a MovePtP movement), you can specify the behavior of the compensating movement.

For the coupling, you can also specify a ratio in the form of a factor and a divisor. A negative factor allows you to adjust the effect of the coupling in the direction of rotation.

The coupling takes the gearbox settings of the axes into account.

ⓘ Info

Condition for creating a coupling group

Coupling to the master axis of the coupling group is only possible with a single follower axis. The functions provide parameters that make it possible to couple several follower axes to one master axis of the coupling group in future extensions.

Creating a coupling group

To use a coupling group, you must first define it as a variable in the STX program:

```
Var
DMKorr: MCCouplingGroup('DruckMKorrektur', AX10, mcMgr);
End_Var;
```

This call creates a new coupling group (`MCCouplingGroup`) using the following parameters:

Parameter	Data type	Meaning
Groupname	String	Freely selectable name.
MasterAxis	Pointer to MCAxis	Axis that is used by "follower axes" as a subordinate axis.
mcMgr	IMCAxisManager	Interface parameter that must be passed in this way.

43.1 Syntax for MCCouplingGroup

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<variable of type MCCouplingGroup>.<function or value access>(optional <pa-
rameter 1, parameter 2, parameter n>);
```

43.2 Functions of MCCouplingGroup

The functionality provides the following functions:

– AddFollowerAxis()

– Redefine()

– Undefine()

43.2.1 AddFollowerAxis()

This function adds a follower axis to the coupling group. The master axis passed in the variable definition is made subordinate to this axis. You also define the ratio for the subordination.

The coupling only takes effect after the Redefine() [▶ 243] function has been called.

Parameters

Parameter	Data type	Meaning
Idx	Integer	Index of the follower axis. Must always be `0`.
AxFollower	Pointer to MCAxis	Object of the follower axis. The master axis is made subordinate to this axis.
Factor	Integer	Factor for the ratio between master and follower axis.
Divisor	Integer	Divisor for the ratio between master and follower axis.

Syntax

```
DMKorr.AddFollowerAxis(0,AX1,1,5);
```

State management

The function can be called in any state of the master and follower axis.

43.2.2 Redefine()

This function makes the coupling created via AddFollowerAxis() [▶ 243] effective.

Parameters

The function has no parameters.

Syntax

```
DMKorr.Redefine();
```

State management

The following table applies to calling the function for the master axis:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

The function can be called in any state of the follower axis.

43.2.3 Undefine()

This function dissolves the coupling group. The axes are decoupled from each other. All internal position offsets are cleared.

Parameters

The function has no parameters.

Syntax

```
DMKorr.Undefine();
```

State management

The following table applies to calling the function for the master and follower axis:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

43.3 Properties of MCCouplingGroup

The functionality provides the following properties:

– FollowerCount
– MasterAxis
– FollowerAxis
– Name

43.3.1 FollowerCount

Description

Data type	Meaning
Integer	Reads the number of defined follower axis objects of the coupling group. This property anticipates future extensions, as only a single follower axis can currently 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

```
NumberOfFollowerAxes := DMKorr.FollowerCount;
```

State management

The property can be read in any state of the master and follower axis.

43.3.2 MasterAxis

Description

Data type	Meaning
Pointer to MCAxis	Reads the axis object that was defined as the master axis in the coupling group.

Syntax

```
MasterAxisObject := DMKorr.MasterAxis;
```

State management

The property can be read in any state of the master and follower axis.

43.3.3 FollowerAxis[]

Description

NOTICE



STX exception if there is no coupling

If you query this property without a follower axis object being present, an exception is generated in the STX program.

► Before querying the property, check whether a coupling is present.

Data type	Meaning
Pointer to MCAxis	Reads the axis object that was defined as the follower axis in the coupling group. The array index is intended for future extensions and must currently always be 0.

Syntax

```
if DMKorr.FollwerAxis[0] != null then
Name_CouplingFollowerAxis := DMKorr.FollowerAxis[0].Info.Name;
end_if
```

State management

The property can be read in any state of the master and follower axis.

43.3.4 Name

Description

Data type	Meaning
String	Reads the name that the coupling group was given in the variable definition.

Syntax

```
CouplingName := DMKorr.Name;
```

State management

The property can be read in any state of the master and follower axis.

44 MCGantryGroup

The **MCGantryGroup** functionality (1:1 mechanically coupled axes) allows you to couple 2 or more mechanically coupled drives that move a load together in such a way that they function like one axis. All axes must have the same mechanical configuration. The following behavior applies:

– If a follower 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 in all axes of the gantry group.

  With every `Enable()` of the gantry group or of the master axis of the gantry group, the actual and setpoint position of the follower axes of the gantry group are set to the value of the actual position of the master axis.

– The deviation from the actual position is monitored.

– Joint homing of all axes involved is possible.

  This is an advantage if one axis alone is not able to move the load.

  Homing is performed via the master axis. At the reference point, the follower axes are loaded with the same reference value as determined by the master axis.

– If the master axis is part of a technology group, the follower axes are treated like the master axis. They do not have to be included in the technology group separately.

You can operate the gantry group either via the object name of the gantry group followed by the designation `MasterAxis` or via the axis object of the master axis. After coupling, direct operation of the follower axis is no longer possible. Actions on the follower axis lead to an error message as long as the coupling is active.

ⓘ Info

Condition for creating a coupling group

Coupling to the master axis of the coupling group is only possible with a single follower axis. The functions provide parameters that make it possible to couple several follower axes to one master axis of the coupling group in future extensions.

Creating a gantry group

To use a gantry group, you must first define it as a variable in the STX program:

```
Var
Portal1: MCGantryGroup('Portal1', AX1, mcMgr);
End_Var;
```

This call creates a new gantry group (`MCGantryGroup`) using the following parameters:

Parameter	Data type	Meaning
Groupname	String	Freely selectable name.
MasterAxis	Pointer to MCAxis	Master axis in the gantry group via which all commands are executed.
mcMgr	IMCAxisManager	Interface parameter that must be passed in this way.

44.1 Syntax for MCGantryGroup

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<variable of type MCGantryGroup>.<function or value access>(optional <param-
eter 1, parameter 2, parameter n>);
```

44.2 Functions of MCGantryGroup

The functionality provides the following functions:

– AddFollowerAxis()
– Redefine()
– Undefine()

44.2.1 AddFollowerAxis()

This function adds a follower axis to the gantry group. The axis then receives the same setpoints as the master axis and can be operated via the master axis.

The coupling only takes effect after the Redefine() [▶ 248] function has been called.

Parameters

Parameter	Data type	Meaning
Idx	Integer	Index of the follower axis.
AxFollower	Pointer to MCAxis	Object of the follower axis.

Syntax

```
DMKorr.AddFollowerAxis(0, AX2);
```

State management

The function can be called in any state of the master and follower axis.

44.2.2 Redefine()

This function makes the coupling created via AddFollowerAxis() [▶ 248] effective.

Parameters

The function has no parameters.

Syntax

```
Portal1.Redefine();
```

State management

The following table applies to calling the function for the **master and follower axis**:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

44.2.3 Undefine()

This function dissolves the gantry group. The axes are decoupled from each other.

Parameters

The function has no parameters.

Syntax

```
Portal1.Undefine();
```

State management

The following table applies to calling the function for the master and follower axis:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

44.3 Properties of MCGantryGroup

The functionality provides the following properties:

– GantryWindow[]
– ConfigGantryWindow[]
– FollowerCount
– MasterAxis
– FollowerAxis[]
– Name

44.3.1 GantryWindow[i]

Description

Data type	Meaning
Double	Reads or writes the maximum permitted position deviation of the follower axis with index `i` from the master axis in the gantry group; specified in the user position unit. Default value: `10,0` If the permissible position deviation is exceeded, the gantry group goes into the error state. The error response is defined with the `ConfigGantryWindow` property.

Syntax

```
Portal1.GantryWindow[0] := 20.0;
GantryWindowValue:= Portal1.GantryWindow[0];
```

State management

The property can be read and written in any state of the follower axis.

44.3.2 ConfigGantryWindow[i]

Description

Data type	Meaning
Integer	Reads or writes the error response of the gantry group if the position deviation of the follower axis with index `i` exceeds the limit value `GantryWindow[i]`; Default value: `5`
1	Only triggers a message. Does not cause any response of the axes.
4	The axes are stopped with the programmed ramp and remain in control after standstill.
5	The axes are stopped with the emergency stop ramp and disabled after standstill (default)
6	The axes are disabled directly.

Syntax

```
Portal1.ConfigGantryWindow[0] := 5;
GantryConfig:= Portal1.ConfigGantryWindow[0];
```

State management

The property can be read and written in any state of the follower axis.

44.3.3 FollowerCount

Description

Data type	Meaning
Integer	Reads the number of defined follower axis objects of the gantry group.

Syntax

```
NumberOfFollowerAxes := Portal1.FollowerCount;
```

State management

The property can be read in any state of the master and follower axis.

44.3.4 MasterAxis

Description

Data type	Meaning
Pointer to MCAxis	Reads the axis object that was defined as the master axis in the gantry group.

Syntax

```
MasterAxisObject := Portal1.MasterAxis;
```

State management

The property can be read in any state of the master and follower axis.

44.3.5 FollowerAxis[i]

Description

NOTICE



STX exception if there is no coupling

If you query this property without a follower axis object being present, an exception is generated in the STX program.

► Before querying the property, check whether a coupling is present.

Data type	Meaning
Pointer to MCAxis	Reads the axis object that was defined in the gantry group as the follower axis with index `i`.

Syntax

```
if Portal.FollwerAxis[0] != null then
Name_GantryFollowerAxis := Portal1.FollowerAxis[0].Info.Name;
end_if
```

State management

The property can be read in any state of the master and follower axis.

44.3.6 Name

Description

Data type	Meaning
String	Reads the name that the gantry group was given in the variable definition.

Syntax

```
Gantryname := Portal1.Name;
```

State management

The property can be read in any state of the master and follower axis.

44.4 Example task MCGantryGroup

Here you will find an example task for a gantry group that contains various Motion API functions:

```
Var
//Definition of a gantry group with name (for the group), master axis
(typically one of the two axes) and MC manager function
    Portal1: MCGantryGroup('Portal1', AX1, mcMgr);
//Additional variables for the example program
    Run: bool;
    Gantry_Count: int;
    GantryWindowValue: double;
    GantryConfig: int;
End_Var;

Task tGantry Autorun

//Dissolving a gantry group - this is normally not necessary, but can be
useful, e.g. if you also want to move the axes individually.
    if Portal1.FollowerCount > 0 then
        Portal1.Undefine();
    end_if;

    //First, a follower axis is added to the group. Several follower axes
can also be added
    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 gantry group.

    Portal1.GantryWindow[0] := 20; //max. following error between master and
follower axis in user units
    GantryWindowValue:= Portal1.GantryWindow[0]; //The value can also be
read back.
    //The permitted following 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 deviation is exceeded, the gantry group goes into 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];

    //Homing
    //Homing is performed with the master axis. When the reference has been
found, the position of the follower axis is also set to this value.
    //With every enable, the actual and setpoint position of the follower
axis are set to those of the master axis.
```

```
    if not Portal1.MasterAxis.State.IsEnabled then
        Portal1.MasterAxis.Power.Enable();
        When Portal1.MasterAxis.State.IsEnabled Continue;
    end_if;
    Portal1.MasterAxis.MoveHome.Start(Directions.Positive,ReferencingSwitchC
onfigs.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();
    When Portal1.MasterAxis.State.IsDisabled Continue

Loop
    //Operation is only performed via the master axis.
    //The setpoints of the master axis are applied to the follower axis/axes
(including an internal offset that is determined when enabling.)
When Run Continue;
    // this is how the gantry is used as an axis

        // With every enable, the actual and setpoint position of the
follower axis are set to those of the master axis.
        if not Portal1.MasterAxis.State.IsEnabled then
            Portal1.MasterAxis.Power.Enable();
            When Portal1.MasterAxis.State.IsEnabled Continue;
        end_if;

        //these commands move both axes together 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();
            When Portal1.MasterAxis.State.IsDisabled Continue;
        end_if;
End_Loop;
End_Task;
```

45 MCGeo.Activate, MCGeo.Deactivate

The **MCGeo.Activate** and **MCGeo.Deactivate** functionalities allow you to activate and deactivate a path group. Changes to a path group are permitted in all operating states of the group except **inactive**.

As soon as a path group has been activated, all member axes are set to the **inactive** operating state. You can then only access these axes via the path group in which they are integrated.

A path group can only be activated and deactivated if all member axes of this group are either in the **enabled** or the **disabled** operating state and all member axes have the same operating state. Otherwise, an error message is output.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

45.1 Syntax for MCGeo.Activate, MCGeo.Deactivate

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Activate();
<group name>.Deactivate();
```

45.2 Functions of MCGeo.Activate, MCGeo.Deactivate

The functionality provides the following functions:

– Activate()
– Deactivate()

45.2.1 Activate()

This function activates a path group, provided that all member axes are either in the **Disabled** or the **Enabled** state.

Parameters

The function has no parameters.

Syntax

```
geoGroup.Activate();
```

State management

The following table applies to calling the function for the master and follower axis:

Initial state	Allowed	Final state
Inactive	√	Disabled/Enabled
Disabled	–	Disabled
Enabled	–	Enabled
Running	–	Running

Initial state	Allowed	Final state
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Query in the program whether the new state has been reached before calling further commands for the axes.

```
When Not geoGroup.State.IsInactive Continue;
```

Alternatively, you can also query the **IsEnabled** or **IsDisabled** operating state of the member axes – depending on which operating state the group had before the command was called.

45.2.2 Deactivate()

This function deactivates a path group. After this command, the group is in the **inactive** state. The member axes of the group can be controlled independently again. They adopt the operating state of the path group.

Parameters

The function has no parameters.

Syntax

```
geoGroup.Deactivate();
```

State management

The following table applies to calling the function for the master and follower axis:

Initial state	Allowed	Final state
Inactive	–	Inactive
Disabled	√	Inactive
Enabled	√	Inactive
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Query in the program whether the new state has been reached before calling further commands for the axes.

```
When geoGroup.State.IsInactive Continue;
```

Alternatively, you can also query the **IsEnabled** or **IsDisabled** operating state of the member axes – depending on which operating state the group had before the command was called.

46 MCGeo.Buffer

The **MCGeo.Buffer** functionality allows you to manage the preliminary buffer of the path group.

Motion commands of the path group are always executed in buffered mode, i.e. they are added to a preliminary buffer and processed one after the other. This also allows blending of movements and programming of entire contours consisting, for example, of several linear and circular movements.

The condition for the transition from one path movement to the next (block change condition) is passed to the preliminary buffer in the motion command via the `MCGeoPathBlockEnd` parameter.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

46.1 Syntax for MCGeo.Buffer

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Buffer.<function>(optional <parameter>);
```

46.2 Functions of MCGeo.Buffer

The functionality provides the following functions:

- Clear()
- Suspend()
- Resume()
- UserIndex.Enable()
- UserIndex.Disable()
- UserIndex.Reset()

46.2.1 Clear()

The function clears the preliminary buffer with the following effects:

- A movement programmed with blending is decelerated along the blending polynomial.
- A movement that has already started is completed.

The commands `Power.Disable()`, `Power.Quickstop()` and `Stop()` also clear the preliminary buffer.

Parameters

This function has no parameters.

Syntax

```
geoGroup.Buffer.Clear();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

46.2.2 Suspend()

The function stops processing of the preliminary buffer at the next block transition with the following effects:

– The current movement is completed.

– The next movement written to the preliminary buffer is no longer started.

– The group remains in *buffered* operating mode.

– A movement programmed with blending is decelerated along the blending polynomial.

As soon as the block during whose processing the `Buffer.Suspend()` function was called has been completed, the path group is in the **Enabled (stopped)** state.

Calling the Resume() [▶ 259] function resumes buffer processing.

Parameters

This function has no parameters.

Syntax

```
geoGroup.Buffer.Suspend();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	–	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

46.2.3 Resume()

The function resumes processing of the preliminary buffer that was stopped with Suspend() [▶ 258].

Parameters

This function has no parameters.

Syntax

```
geoGroup.Buffer.Resume();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	–	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

46.2.4 UserIndex.Enable()

The function activates the path segment counter function. The path segment counter is incremented when a programmed path movement of the path group has been started. If blending is performed between two path movements, the path segment counter is incremented after blending has been completed.

It can be reset to zero with the UserIndex.Reset() [▶ 260] command.

Parameters

This function has no parameters.

Syntax

```
geoGroup.Buffer.UserIndex.Enable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled

Initial state	Allowed	Final state
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

46.2.5 UserIndex.Disable()

The function deactivates the path segment counter function.

Parameters

This function has no parameters.

Syntax

```
geoGroup.Buffer.UserIndex.Disable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

46.2.6 UserIndex.Reset()

The function resets the path segment counter to zero. The reset also enables the path segment counter.

Parameters

This function has no parameters.

Syntax

```
geoGroup.Buffer.UserIndex.Reset();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

46.3 Properties of MCGeo.Buffer

The functionality provides the following properties:

– UserIndex.Actual

46.3.1 UserIndex.Actual

Description

Data type	Meaning
Integer	Reads the number of path segments traveled since the path segment counter was last reset.

Syntax

```
NumberOfPathSegments := geoGroup.Buffer.UserIndex.Actual;
```

State management

The property can be read in any state.

47 MCGeo.Coordinates

The **MCGeo.Coordinates** functionality allows you to query the current values of the path movement sorted by coordinates.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

47.1 Syntax for MCGeo.Coordinates

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Coordinates.<value access>;
```

47.2 Properties of MCGeo.Coordinates

The functionality provides the following properties:

– Actual
– Setpoint
– Target

47.2.1 Actual

Description

This property allows you to read the current 6D coordinate points of the path group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Actual	Coordinates6D	Actual value of the TCP position in 6D spatial coordinates.
Actual.X	Coordinates6D.X (Double)	Current value of the TCP position in X direction.
Actual.Y	Coordinates6D.Y (Double)	Current value of the TCP position in Y direction.
Actual.Z	Coordinates6D.Z (Double)	Current value of the TCP position in Z direction.
Actual.U	Coordinates6D.U (Double)	Current value of the TCP orientation in U direction.
Actual.V	Coordinates6D.V (Double)	Current value of the TCP orientation in V direction.
Actual.W	Coordinates6D.W (Double)	Current value of the TCP orientation in W direction.

Syntax

```
Actual6D := geoGroup.Coordinates.Actual;
ActualX := geoGroup.Coordinates.Actual.X;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

47.2.2 Setpoint

Description

This property allows you to read the current 6D coordinate points of the setpoint generator of the path group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Setpoint	Coordinates6D	Setpoint of the TCP position in 6D spatial coordinates.
Setpoint.X	Coordinates6D.X (Double)	Setpoint of the TCP position in X direction.
Setpoint.Y	Coordinates6D.Y (Double)	Setpoint of the TCP position in Y direction.
Setpoint.Z	Coordinates6D.Z (Double)	Setpoint of the TCP position in Z direction.
Setpoint.U	Coordinates6D.U (Double)	Setpoint of the TCP orientation in U direction.
Setpoint.V	Coordinates6D.V (Double)	Setpoint of the TCP orientation in V direction.
Setpoint.W	Coordinates6D.W (Double)	Setpoint of the TCP orientation in W direction.

Syntax

```
Setpoint6D := geoGroup.Coordinates.Setpoint;
SetpointX := geoGroup.Coordinates.Setpoint.X;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

47.2.3 Target

Description

This property allows you to read the current 6D coordinate points of the target of the current path movement of the path group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Target	Coordinates6D	Target of the TCP position in 6D spatial coordinates.
Target.X	Coordinates6D.X (Double)	Target of the TCP position in X direction.
Target.Y	Coordinates6D.Y (Double)	Target of the TCP position in Y direction.
Target.Z	Coordinates6D.Z (Double)	Target of the TCP position in Z direction.
Target.U	Coordinates6D.U (Double)	Target of the TCP orientation in U direction.
Target.V	Coordinates6D.V (Double)	Target of the TCP orientation in V direction.
Target.W	Coordinates6D.W (Double)	Target of the TCP orientation in W direction.

Syntax

```
Target6D := geoGroup.Coordinates.Target;
TargetX := geoGroup.Coordinates.Target.X;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

48 MCGeo.Coupling

The **MCGeo.Coupling** functionality allows you to control the tool center point movement of a path group using the position of up to six other independent MCX axes. The path group is coupled to the other axis/axes and follows their movement. For example, the gripper arm of a multi-axis robot can be coupled to the movement of a conveyor belt from which it is to pick something up.

Coupling types

There are 2 ways of controlling the movement:

1. The path movement follows the MCX axes in a linear movement.
2. The path movement follows the MCX axes in the form of a circular movement.

You can couple the path group to up to six MCX axes. Depending on the selected coupling type, 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 X direction follows the first axis specified in the coupling command linearly.
– The movement of the path group in Y direction follows the second axis specified in the coupling command linearly.
– The movement of the path group in Z direction follows the third axis specified in the coupling command linearly.
– The U orientation of the path group follows the fourth axis specified in the coupling command linearly.
– The V orientation of the path group follows the fifth axis specified in the coupling command linearly.
– The W orientation of the path group follows the sixth axis specified in the coupling command linearly.

Motion control for coupling type 2: circular path movement:

This coupling type additionally requires the center point coordinates of the circle and the coordinate system plane in which the circle is to lie.

The first 3 coupled axes specify the circular movement in polar coordinates.

– The radius of the circular movement of the path group follows the first axis specified in the coupling command.
– The angle of the circular movement of the path group follows the second axis specified in the coupling command.
– The helix of the circular movement of the path group follows the third axis specified in the coupling command linearly.
– The U orientation of the path group follows the fourth axis specified in the coupling command linearly.
– The V orientation of the path group follows the fifth axis specified in the coupling command linearly.
– The W orientation of the path group follows the sixth axis specified in the coupling command linearly.

ⓘ Info

Further information

For procedures and sample projects on working with synchronous axes in path groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group* and *Path group – synchronous axis* . These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

48.1 Syntax for MCGeo.Coupling

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Coupling.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

48.2 Functions of MCGeo.Coupling

The functionality provides the following functions:

– Couple()
– Decouple()
– GetPositioningMode()
– GetCoupleType()
– GetCoupleMode()
– GetDecoupleMode()

48.2.1 Couple()

This function configures the coupling function between the MCX axes and the path group and establishes the coupling. The process requires the path group to be in the **Enabled** operating state. In the coupled state, the operating state of the path group is always **Running** and it can no longer be moved via the `Move` commands.

Parameters

Parameter	Data type	Meaning
PositioningMode	MCGeoPositioning-Modes (optional)	Defines how the group follows the positions of the MCX axes.
	MCGeoPositioning-Modes.Absolute	The group follows the position of the MCX axes relative to the same values of the setpoint position and the offsets: *Setpoint position of the TCP coordinate = setpoint position of the MCX axis + offset*. At the start of coupling, the position values of the TCP and of the MCX axes involved (shifted by the offsets) must match in order to avoid a jerky movement of the TCP.
	MCGeoPositioning-Modes.Relative	The group follows the position of the MCX axes plus its starting positions and the offsets of the MCX axes: *Setpoint position of the TCP coordinate = setpoint position of the MCX axis + starting position of the TCP coordinate + offset*

Parameter	Data type	Meaning
CoupleType	MCGeoCoupleTypes (optional)	Type of TCP movement depending on the MCX axes.
	MCGeoCouple-Types.Linear	The setpoint positions of the MCX axes serve directly as the specification for the setpoint positions of the path coordinates.
	MCGeoCouple-Types.CircularXY	The setpoint positions of the MCX axes serve as the specification for the polar coordinates (see above) of a circular movement in the XY plane.
	MCGeoCouple-Types.CircularYZ	The setpoint positions of the MCX axes serve as the specification for the polar coordinates (see above) of a circular movement in the YZ plane.
	MCGeoCoupleTypesCircularZX	The setpoint positions of the MCX axes serve as the specification for the polar coordinates (see above) of a circular movement in the ZX plane.
CoupleMode	MCGeoCouple-Modes.Immediate (optional)	The coupling is executed immediately.
CoupleID_1	Integer (optional)	Object ID of the first MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 110]).
CoupleID_2	Integer (optional)	Object ID of the second MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 110]).
CoupleID_3	Integer (optional)	Object ID of the third MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 110]).
CoupleID_4	Integer (optional)	Object ID of the fourth MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 110]).
CoupleID_5	Integer (optional)	Object ID of the fifth MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 110]).
CoupleID_6	Integer (optional)	Object ID of the sixth MCX axis to which the path group is to be coupled (see MCAxis.Info [▶ 110]).
Offset_1	Double (optional)	Offset between the setpoint position of the first MCX axis and the setpoint for the corresponding path coordinate.
Offset_2	Double (optional)	Offset between the setpoint position of the second MCX axis and the setpoint for the corresponding path coordinate.
Offset_3	Double (optional)	Offset between the setpoint position of the third MCX axis and the setpoint for the corresponding path coordinate.
Offset_4	Double (optional)	Offset between the setpoint position of the fourth MCX axis and the setpoint for the corresponding path coordinate.
Offset_5	Double (optional)	Offset between the setpoint position of the fifth MCX axis and the setpoint for the corresponding path coordinate.
Offset_6	Double (optional)	Offset between the setpoint position of the sixth MCX axis and the setpoint for the corresponding path coordinate.
CenterPointX	Double (optional)	X coordinate of the circle center point if `CircularXY` or `CircularZX` was selected as `CoupleType`.
CenterPointY	Double (optional)	Y coordinate of the circle center point if `CircularXY` or `CircularYZ` was selected as `CoupleType`.

Parameter	Data type	Meaning
CenterPointZ	Double (optional)	Z coordinate of the circle center point if `CircularYZ` or `CircularZX` was selected as CoupleType.

Syntax

```
geoGroup.Coupling.Couple(MCGeoPositioningModes.Absolute,MCGeoCouple-
Types.Linear,MCGeoCoupleModes.Immediate,AX1.Info.ObjectID,,,,,,10.0,,,,,,);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Running
Running	–	Running
Suspended	–	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

48.2.2 Decouple()

Parameters

Parameter	Data type	Meaning
Mode	MCGeoDecoupleModes.Immediate (optional)	Decouples the path group from the independent MCX axes to which the path group was coupled.

Syntax

```
geoGroup.Coupling.Decouple(MCGeoDecoupleModes.Immediate);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	–	Enabled
Running	√	Enabled
Suspended	–	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

48.2.3 GetPositioningMode()

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the current positioning mode of the coupled path group. The return value is of type `MCGeoPositioningModes`.

Syntax

```
CouplePM := geoGroup.Coupling.GetPositioningMode(AX1);
```

State management

The function can be read in any state.

48.2.4 GetCoupleType()

Parameters

Parameter	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 management

The function can be read in any state.

48.2.5 GetCoupleMode()

Parameters

Parameter	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 management

The function can be read in any state.

48.2.6 GetDecoupleMode()

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the current decoupling mode of the coupled path group. The return value is of type `MCGeoDecoupleModes`**. It is currently always** `Immediate` **(**`1`**).**

Syntax

```
AuskopDecoupleM := geoGroup.Coupling.GetDecoupleMode(AX1);
```

State management

The function can be read in any state.

48.3 Properties of MCGeo.Coupling

The functionality provides the following properties:

– CoupleState

48.3.1 CoupleState

Description

Property	Data type	Meaning
CoupleState	MCGeoCoupleStates	Reads the coupling status of the path group.
	MCGeoCoupleStates.Decoupled	The path group is not coupled to other axes.
	MCGeoCoupleStates.Coupled	The path group is in the coupled state.

Syntax

```
CoupleStatus := geoGroup.Coupling.CoupleState;
```

State management

The property can be read in any state.

49 MCGeo.Diagnostics

The **MCGeo.Diagnostics** functionality allows you to manage the error handling of a path group.

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

The meaning of the error and message numbers is listed in the JetSym online help under "MotionControl errors".

49.1 Syntax for MCGeo.Diagnostics

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Diagnostics.<function or value access>(optional <parameter>);
```

49.2 Functions of MCGeo.Diagnostics

The functionality provides the following functions:

– ClearErrors()
– Software.ClearErrors()
– Software.GetErrorCode()
– Software.GetMessageCode()

49.2.1 ClearErrors(), Software.ClearErrors() – path group

For errors, the functions acknowledge an error state in the path group. The entries in the error buffer are cleared.

Messages do not result in an error response and therefore do not need to be acknowledged. For messages, both functions clear the entries in the message buffer.

Parameters

These functions have no parameters.

Syntax

```
geoGroup.Diagnostics.ClearErrors();
geoGroup.Diagnostics.Software.ClearErrors();
```

ⓘ Info

Scope of the command

These commands clear all errors assigned to the group. These can be errors triggered by a group object as well as errors reported by the composite axes.

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	–	Enabled

Initial state	Allowed	Final state	
Running	–	Running	
Suspended	–	Suspended	
Error: enabled	√	Enabled	The system only changes to this state when the cause of the error has been eliminated. Otherwise, the error state may persist or be reached again immediately.
Error: disabled	√	Disabled	

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

49.2.2 Software.GetErrorCode()

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

Parameters

Parameter	Data type	Meaning
BufferIndex	Integer (optional)	`BufferIndex` passes the position in the error buffer whose error number is to be read. The maximum value of the `BufferIndex` parameter can be read via the `ErrorCount` property.

Syntax

```
geoGroup.Diagnostics.Software.GetErrorCode();
```

State management

The function can be read in any state.

49.2.3 Software.GetMessageCode()

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

Parameters

Parameter	Data type	Meaning
BufferIndex	Integer (optional)	`BufferIndex` passes the position in the message buffer whose message number is to be read. The maximum value of the `BufferIndex` parameter can be read via the `MessageCount` property.

Syntax

```
geoGroup.Diagnostics.Software.GetMessageCode();
```

State management

The function can be read in any state.

49.3 Properties of MCGeo.Diagnostics

The functionality provides 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 recorded since the `ClearErrors()` or `Software.ClearErrors()` function was last executed, or since the control unit was restarted.

Syntax

```
NumberOfErrors := geoGroup.Diagnostics.Software.ErrorCount;
```

State management

The property can be read in any state.

49.3.2 IsErrorPending, Software.IsErrorPending

Description

Data type	Meaning
Bool	Queries whether an error is pending in the MCX system. This is equivalent to comparing `ErrorCount` with `0`.

Syntax

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

State management

The property can be queried in any state.

49.3.3 Software.MessageCount

Description

Data type	Meaning
Integer	Reads the number of messages recorded since the `ClearErrors()` or `Software.ClearErrors()` function was last executed, or since the control unit was restarted.

Syntax

```
NumberOfMessages := geoGroup.Diagnostics.Software.MessageCount;
```

State management

The property can be queried in any state.

49.3.4 IsMessagePending, Software.IsMessagePending

Description

Data type	Meaning
Bool	Queries whether a message is pending in the MCX system. This is equivalent to comparing `MessageCount` with `0`.

Syntax

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

State management

The property can be queried in any state.

49.4 Example task for MCGeo.Diagnostics

Here you will find an example task for a single axis that contains 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 the error variables used for display
    For i := 0 To 9 Do
        GeoError[i] := 0;
    End_For;
    AcknowledgeError := False;

    When geoGroup.Diagnostics.IsErrorPending Continue;

        Taskbreak Automatic;
        // insert further error handling here

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

    // Errors have been eliminated
    When AcknowledgeError Continue;
```

```
        geoGroup.Diagnostics.ClearErrors();
        Taskrestart Automatic;
End_Loop;

End_Task;
```

50 MCGeo.Events

The **MCGeo.Events** functionality allows you to manage the behavior of flags that are set depending on the progress of the movement in the preliminary buffer of the path group.

ⓘ Info

`SetFlag()` function in connection with the preliminary buffer

The `Event` commands are also transferred to the preliminary buffer. A condition for setting a flag (`SetFlag()` function) must be written to the preliminary buffer before the path segment to which it refers.

Working with the preliminary buffer makes it possible to travel along path contours that consist of several path segments and/or merge into each other by blending. The functions of the **Events** functionality allow you to have flags set by the MCX kernel. The flags depend on which path segment in the preliminary buffer is currently being traveled and where the movement in the path segment currently is. These flags in turn allow you to control actions in the STX program, e.g. opening a dosing valve.

When the path group is activated, all flags are cleared.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

50.1 Syntax for MCGeo.Events

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Events.<function or value access>(optional <parameter>);
```

50.2 Functions of MCGeo.Events

The functionality provides the following functions:

- SetFlag()
- ReadFlags()
- ReadFlag()
- ClearFlags()

50.2.1 SetFlag()

This function causes the MCX kernel to set a flag depending on the parameter passed when executing the following path segment in the preliminary buffer.

If the condition at the end of the movement is selected for a movement with blending, the flag is set at the beginning of the blending polynomial.

Parameters

Parameter	Data type	Meaning
BlockCondition	MCGeoEventConditions	Condition that defines at which point of the movement the flag with the `EventNo` is set.
	MCGeoEventConditions.BlockStart	The flag is set at the start of the movement.
	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 has been reached.
	MCGeoEventConditions.BlockEnd_InPosState_DestWnd_Delay	The flag is set at the end of the movement and after the programmed waiting time.
EventNo	Integer (optional)	Number of the flag (1 ... 16) to be set under the above condition.

Syntax

```
geoGroup.Events.SetFlag(MCGeoEventConditions.BlockStart,2);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	X	Disabled
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

50.2.2 ReadFlag()

This function allows you to read the status of a specific event flag.

Parameters

Parameter	Data type	Meaning
EventNo	Integer	Number of the flag (1 ... 16) whose status is to be read.

Syntax

```
If geoGroup.Events.ReadFlag(2) then;
```

State management

The function can be called in any state.

50.2.3 ReadFlags()

This function allows you to read the status of all 16 event flags. The function returns a value of data type Integer. The last 16 bits of this value correspond to the flags with the numbers 1 … 16. (Bit 0 = EventFlag 1, ...).

Parameters

The function has no parameters.

Syntax

```
FlagWord := geoGroup.Events.ReadFlags();
```

State management

The function can be called in any state.

50.2.4 ClearFlags()

This function clears the flags of the path group passed in the event mask.

Clearing the flags is not stored in the preliminary buffer, but executed immediately.

Parameters

Parameter	Data type	Meaning
EventMask	Integer	Bits 0 ... 15 are assigned to flags 1 ... 16. A set bit clears the corresponding event flag.

Syntax

```
geoGroup.Events.ClearFlags(0x0001);
```

State management

The following table applies to writing the function:

State monitoring

If the `ClearFlags()` command is immediately followed by a `Deactivate()` command, the corresponding flags may not be cleared, as the `Deactivate()` command has a higher priority if it is processed in the same MCX cycle. However, since the `ClearFlags()` function is not executed in the **inactive** state, in this case it makes sense to query whether the corresponding flags have been cleared before the `Deactivate()` command.

50.3 Example task MCGeo.Events

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task PathMovement Autorun
...
    //Start three linear movements - EventFlag 2 is to be set at the
beginning of the second movement
    geoGroup.Power.Enable();
    When geoGroup.State.IsEnabled Continue;
    geoGroup.Mechanism.Limitations.MaxPathAcceleration := 12000;
    geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute,MCGeoPathBlockE
nd.InPosState,300,20,0,0.1,100,50);
```

```
geoGroup.Events.SetFlag(MCGeoEventConditions.BlockStart,2);
    geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute,MCGeoPathBlockE
nd.InPosState,300,10,500,0.1,150,150);
    geoGroup.MoveLinear.Start(MCGeoPositioningModes.Absolute,MCGeoPathBlockE
nd.InPosState,300,10,0,0.1,0,0);
...
End_Task;


Task Dosing Autorun

    If geoGroup.Events.ReadFlag(2) Then
        oDosing := true; //Switch on dosing at the beginning of the second
path segment
    End_if;
...
End_Task;
```

51 MCGeo.Info

The **MCGeo.Info** functionality allows you to identify the group objects of the motion control.

This is information that the user has defined in the hardware manager and motion setup in the project.

51.1 Syntax for MCGeo.Info

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Info.<property>;
```

51.2 Properties of the functionality

The functionality provides the following properties:

– Name
– ObjectID
– ClassID

51.2.1 Name

Description

Data type	Meaning
String	Queries the name of the path 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

```
GeoName := Geo.Info.Name;
```

In this case, `GeoName` contains the string `Geo`.

State management

The property can be read in any state.

51.2.2 ObjectID

Description

Data type	Meaning
Integer	Queries the controller-internal number of the path group. The property is normally only required in connection with applications that use the functions of the MC manager. A possible application is described in the corresponding section MCManager (MCMgr) [▶ 402].

Syntax

```
GroupID := geoGroup.Info.ObjectID;
```

State management

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. The property is normally only required in connection with applications that use the functions of the MC manager. A possible application is described in the corresponding section MCManager (MCMgr) [▶ 402].

Syntax

```
ClassID := geoGroup.Info.ClassID;
```

State management

The property can be read in any state.

51.3 Example task MCGeo.Info

Here you will find an example task for a single axis that contains various Motion API functions:

After the example task has been executed, the trace shows for all path groups whether or not they are in the **running** state. The groups are listed with 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 by 1 do
        TraceRunningState(mcMgr.GeoList[nGeoIndex]);
    end_for;
end_task;
```

52 MCGeo.Mechanism.Kinematics

The **MCAxis.Mechanism.Kinematics** functionality allows you to read the mechanical design of a path group and to configure it in exceptional cases. The configuration is usually made in the motion setup during commissioning.

The control unit must know the mechanical design of a path group so that the units and the kinematics are calculated correctly in the MCX kernel. This is done in JetSym using the definition in the hardware tree and the motion setup. This data is saved in the file *MCGeo.ini* for the group and loaded into the control unit. During booting, this information is read from the ini file and is known when the application program starts.

Special case: changing the kinematics configuration with `Configure`

For rare exceptional cases, the kinematics configuration can be changed via the `Configure` subfunctions. Note that in this case the mechanical parameters known in the JetSym project are not used in the machine. This can lead to confusion, e.g. during service work. Since these mechanical parameters do not normally change in a machine during operation, it also makes no sense to adjust them while the application program is running.

52.1 Syntax for MCGeo.Mechanism.Kinematics

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Mechanism.Kinematics.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

52.2 Properties of MCGeo.Mechanism.Kinematics

The functionality provides the following properties of the `Mechanism` function:

– Cartesian2D
– Cartesian3D
– Scara

52.2.1 Cartesian2D

Description

The `Offset` and `InvertDirection` properties allow you to read the currently valid kinematic parameters of the coordinates of the path group.

Property	Data type	Meaning
X.Offset	Double	Shift of the linear position in X direction.
X.InvertDirection	Bool	Counting direction reversal of the linear position in X direction.
Y.Offset	Double	Shift of the linear position in Y direction.
Y.InvertDirection	Bool	Counting direction reversal of the linear position in Y direction.

Syntax

```
XOffset := geoGroup.Mechanism.Kinematics.Cartesian2D.X.Offset;
```

State management

The property can be read in any state.

52.2.2 Cartesian3D

Description

The `Offset` and `InvertDirection` properties allow you to read the currently valid kinematic parameters of the coordinates of the path group.

Property	Data type	Meaning
X.Offset	Double	Shift of the linear position in X direction.
X.InvertDirection	Bool	Counting direction reversal of the linear position in X direction.
Y.Offset	Double	Shift of the linear position in Y direction.
Y.InvertDirection	Bool	Counting direction reversal of the linear position in Y direction.
Z.Offset	Double	Shift of the linear position in Z direction.
Z.InvertDirection	Bool	Counting direction reversal of the linear position in Z direction.

Syntax

```
XOffset := geoGroup.Mechanism.Kinematics.Cartesian3D.X.Offset;
```

State management

The property can be read in any state.

52.2.3 Scara

The following figure schematically shows the basic terms of a path kinematics using the example of a SCARA robot.

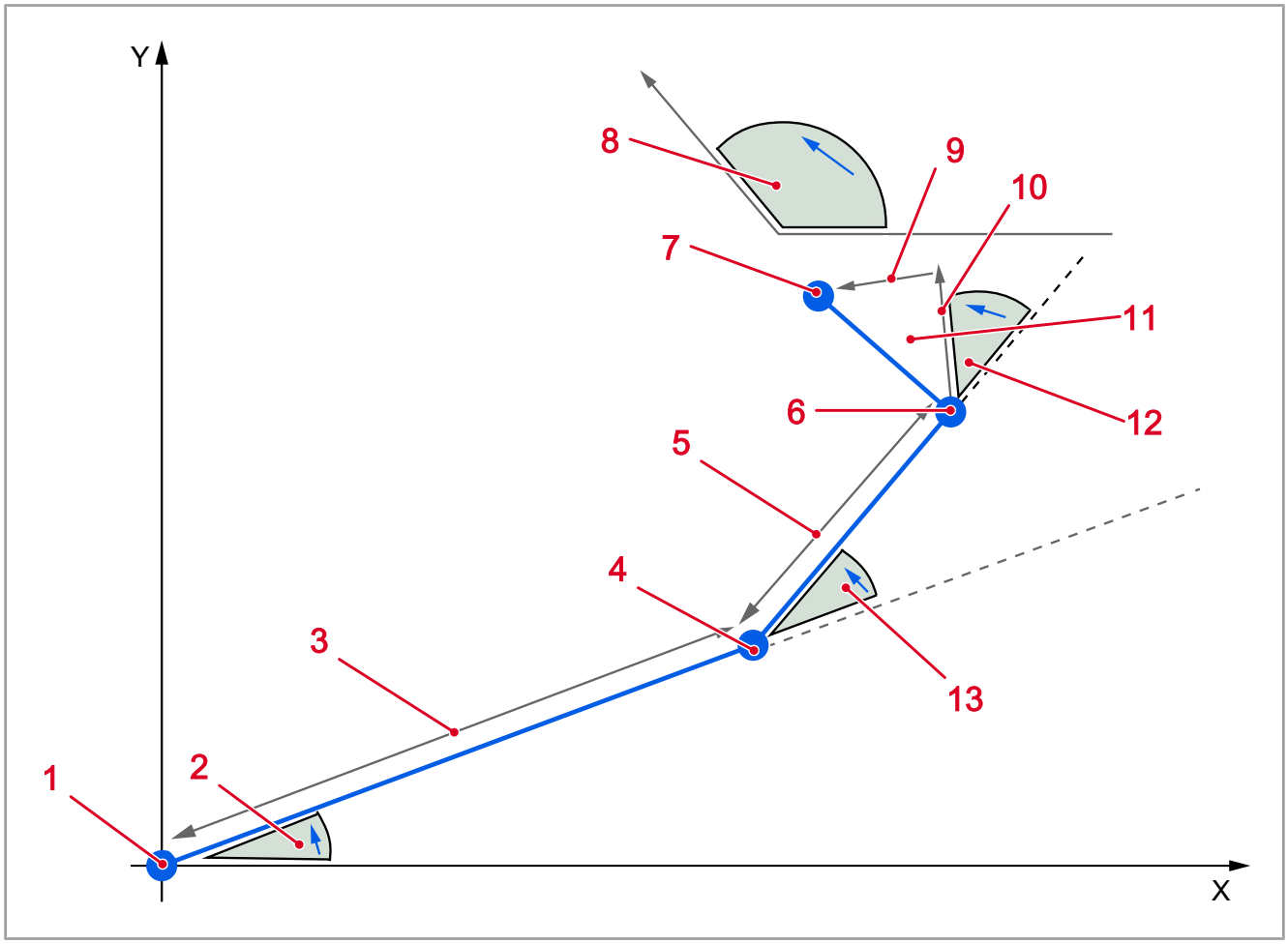

Fig. 9: SCARA kinematics

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

The `Offset` and `InvertDirection` properties allow you to read the currently valid kinematic parameters of the coordinates of the path group.

Property	Data type	Meaning
Shoulder.Offset	Double	Shift of the position of the shoulder axis.
Shoulder.InvertDirection	Bool	Counting direction reversal of the position of the shoulder axis.
Shoulder.ArmLength	Double	Arm length of the shoulder axis (upper arm).
Ellbow.Offset	Double	Shift of the position of the elbow axis.
Ellbow.InvertDirection	Bool	Counting direction reversal of the position of the elbow axis.
Ellbow.ArmLength	Double	Arm length of the elbow axis (forearm).
Vertical.Offset	Double	Shift of the position of the vertical axis.
Vertical.InvertDirection	Bool	Counting direction reversal of the position of the vertical axis.
Wrist.Offset	Double	Shift of the position of the wrist axis.
Wrist.InvertDirection	Bool	Counting direction reversal of the position of the wrist axis.
Wrist.TcpOffsetX	Double	Shift of the tool center point from the center of the wrist axis in X direction.
Wrist.TcpOffsetY	Double	Shift of the tool center point from the center of the wrist axis in Y direction.

Syntax

```
WristOffset := geoGroup.Mechanism.Kinematics.Scara.Wrist.Offset;
```

State management

The property can be read in any state.

52.3 Configuration functions of MCGeo.Mechanism.Kinematics

For the rare exceptional cases mentioned, you can configure the kinematic parameters from the STX program. The subfunction "Configure" is available for this purpose in each case:

- Cartesian2D.X.Configure(), Cartesian2D.Y.Configure()
- Cartesian3D.X.Configure(),Cartesian3D.Y.Configure()
- Scara.Shoulder.Configure(), Scara.Ellbow.Configure()
- Scara.Vertical.Configure()
- Scara.Wrist.Configure()

52.3.1 Cartesian2D.X.Configure(),Cartesian2D.Y.Configure()

The `Configure()` subfunction passes the kinematic parameters for the kinematics definition of a coordinate axis of the path group.

If no parameter is passed, the value passed last is used.

Parameters

Parameter	Data type	Meaning
Offset	Double (optional)	Shift of the linear position.
InvertDirection	Modes (optional)	Counting direction reversal of the linear position.
	Modes.On	Activates the counting direction reversal.
	Modes.Off	Deactivates the counting direction reversal.

Syntax

```
geoGroup.Mechanism.Kinematics.Cartesian2D.X.Configure(1200.0,Modes.Off);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

52.3.2 Cartesian3D.X.Configure(), Cartesian3D.Y.Configure()

The `Configure()` subfunction passes the kinematic parameters for the kinematics definition of a coordinate axis of the path group.

If no parameter is passed, the value passed last is used.

Parameters

Parameter	Data type	Meaning
Offset	Double (optional)	Shift of the linear position.
InvertDirection	Modes (optional)	Counting direction reversal of the linear position.
	Modes.On	Activates the counting direction reversal.
	Modes.Off	Deactivates the counting direction reversal.

Syntax

```
geoGroup.Mechanism.Kinematics.Cartesian3D.X.Configure(1200.0,Modes.Off);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

52.3.3 Scara.Shoulder.Configure(), Scara.Ellbow.Configure()

The `Configure()` subfunction passes the kinematic parameters for the kinematics definition of a coordinate axis of the path group.

Parameters

Parameter	Data type	Meaning
Offset	Double (optional)	Shift of the linear position.
InvertDirection	Modes (optional)	Counting direction reversal of the linear position.
	Modes.On	Activates the counting direction reversal.
	Modes.Off	Deactivates the counting direction reversal.
ArmLength	Double (optional)	Arm length of the shoulder/elbow axis.

Syntax

```
geoGroup.Mechanism.Kinematics.Scare.Shoulder.Config-
ure(10.0,Modes.Off,205.5);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

52.3.4 Scara.Vertical.Configure()

The `Configure()` subfunction passes the kinematic parameters for the kinematics definition of a coordinate axis of the path group.

Parameters

Parameter	Data type	Meaning
Offset	Double (optional)	Shift of the linear position.
InvertDirection	Modes (optional)	Counting direction reversal of the linear position.
	Modes.On	Activates the counting direction reversal.
	Modes.Off	Deactivates the counting direction reversal.

Syntax

```
geoGroup.Mechanism.Kinematics.Scare.Vertical.Configure(25.0,Modes.Off);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

52.3.5 Scara.Wrist.Configure()

The `Configure()` subfunction passes the kinematic parameters for the kinematics definition of a coordinate axis of the path group.

Parameters

Parameter	Data type	Meaning
Offset	Double (optional)	Shift of the linear position.
InvertDirection	Modes (optional)	Counting direction reversal of the linear position.
	Modes.On	Activates the counting direction reversal.
	Modes.Off	Deactivates the counting direction reversal.
TcpOffsetX	Double (optional)	Shift of the tool center point from the center of the wrist axis in X direction.
TcpOffsetY	Double (optional)	Shift of the tool center point from the center of the wrist axis in Y direction.

Syntax

```
geoGroup.Mechanism.Kinematics.Scare.Wrist.Config-
ure(10.0,Modes.Off,50.0,50.0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

53 MCGeo.Mechanism.Limitations

The **MCGeo.Mechanism.Limitations** functionality allows you to pass the mechanical conditions of the path group in the machine to the MCX control. These are used as limit values for motion programming.

If the values for the workspace limitation are exceeded by the programming of a movement, an error is output and the movement is not started.

When the motion profile is calculated, the maximum values for speed, acceleration and, in the case of jerk-limited motion types, also for jerk are taken into account. If the motion command results in higher values for speed, acceleration and jerk, these are automatically limited to the maximum values without an error message.

To illustrate the kinematic variables, the following figure shows a schematic mechanical design using the example of a SCARA robot.

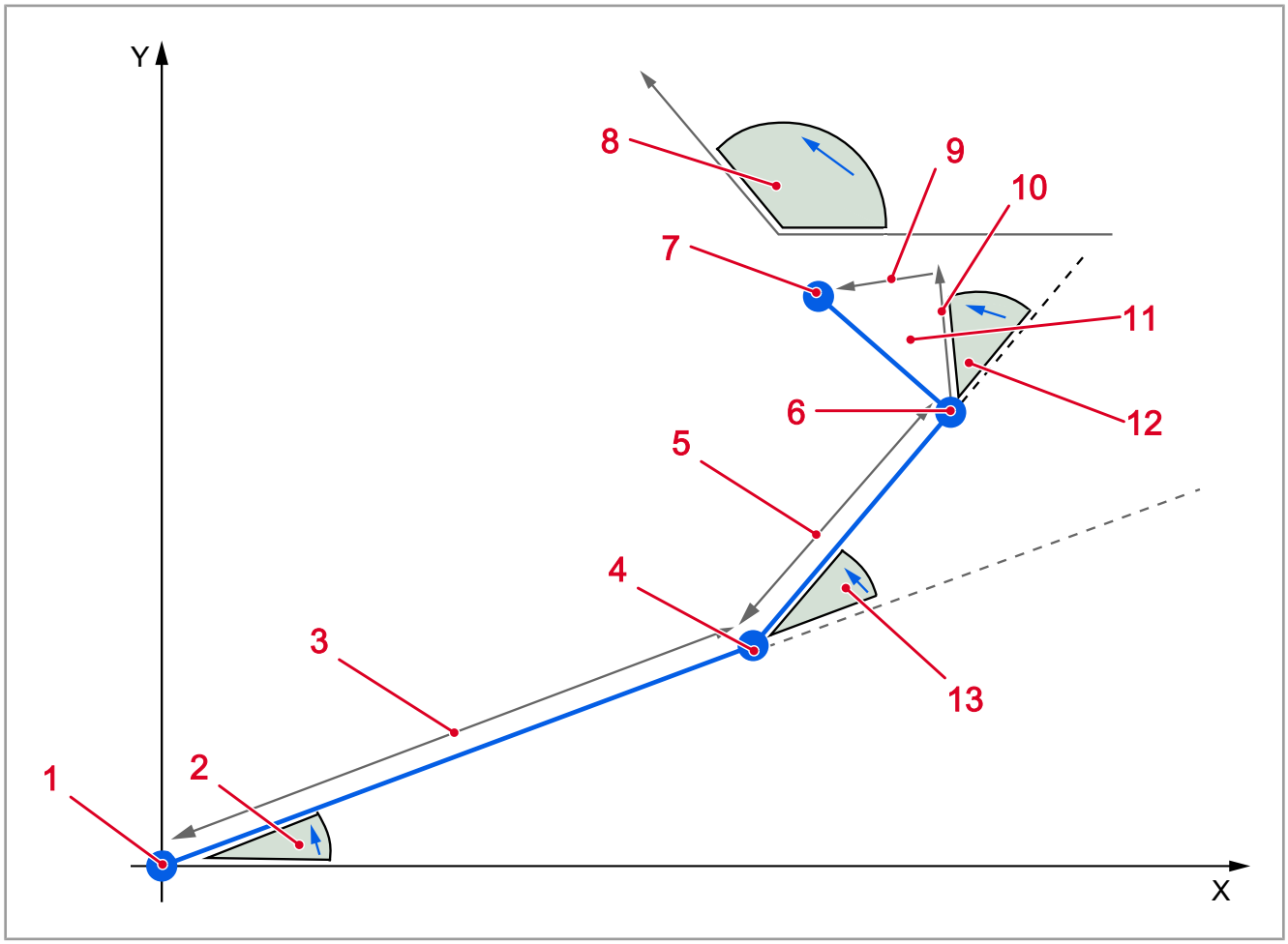

Fig. 10: SCARA kinematics

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 described by the so-called **cardan angles**, in the rotation axis sequence ***z – y‘ – x“***, with respect to the base coordinate system.

Determining the cardan angles

To derive the coordinate system of the TCP ***(x‘“, y‘“, z‘“)*** from the base coordinate system (x, y, z), the following scheme is applied:

1. First, the fixed coordinate system is rotated by the angle U about the ***z*** axis. This results in the coordinate system ***(x‘, y‘, z‘)***.
2. This is now rotated by the angle V about the ***y‘*** axis, resulting in the coordinate system ***(x“, y“, z“)***.
3. Finally, this coordinate system is rotated by the angle W about the ***x“*** axis. This results in the coordinate system of the TCP ***(x‘“, y‘“, z‘“)***.

Meaning of the cardan angles

– The angle U (also yaw angle) describes the alignment of the TCP with respect to the ***x-y*** plane of the base coordinate system.
– The angle V (also pitch angle) describes the alignment of the TCP with respect to the ***y-z*** plane of the base coordinate system.
– The angle W (also roll angle) describes the rotation of the TCP about the ***x*** axis of the TCP coordinate system.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

53.1 Syntax for MCGeo.Mechanism.Limitations

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Mechanism.Limitations.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

53.2 Properties of MCGeo.Mechanism.Limitations

The functionality provides 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 group coordinates X, Y and Z or via a variable of type `Position3D`.

Property	Data type	Meaning
WorkspaceLimitMin	Position3D	Smallest position values for the spatial coordinates in the workspace; absolute value relative to the reference point.
WorkspaceLimitMin.X	Double	Smallest position value for the spatial coordinate X in the workspace.
WorkspaceLimitMin.Y	Double	Smallest position value for the spatial coordinate Y in the workspace.
WorkspaceLimitMin.Z	Double	Smallest position value for the spatial coordinate Z in the workspace.

Syntax

```
ArbeitsbereichMin := geoGroup.Mechanism.Limitations.WorkspaceLimitMin;
geoGroup.Mechanism.Limitations.WorkspaceLimitMin := -100.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

53.2.2 WorkspaceLimitMax

Description

The property reads or writes the largest value for the workspace coordinates of the path group. The values are passed and read either individually for the group coordinates X, Y and Z or via a variable of type `Position3D`.

Property	Data type	Meaning
WorkspaceLimitMax	Position3D	Largest position values for the spatial coordinates in the workspace; absolute value relative to the reference point.
WorkspaceLimitMax.X	Double	Largest position value for the spatial coordinate X in the workspace.
WorkspaceLimitMax.Y	Double	Largest position value for the spatial coordinate Y in the workspace.
WorkspaceLimitMax.Z	Double	Largest position value for the spatial coordinate Z in the workspace.

Syntax

```
ArbeitsbereichMax := geoGroup.Mechanism.Limitations.WorkspaceLimitMax;
geoGroup.Mechanism.Limitations.WorkspaceLimitMax := 1000.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

53.2.3 OrientationLimitMin

Description

The property reads or writes the smallest value for the orientation angles in the workspace of the path group. The values are passed and read either individually for the group orientations U, V and W or via a variable of type `Position3D`.

Property	Data type	Meaning
OrientationLimitMin	Position3D	Smallest orientation angle set (cardan angles U, V, W).

Property	Data type	Meaning
OrientationLimitMin.U	Double	Smallest orientation angle U.
OrientationLimitMin.V	Double	Smallest orientation angle V.
OrientationLimitMin.W	Double	Smallest orientation angle W.

Syntax

```
OrientationRangeMin := geoGroup.Mechanism.Limitations.OrientationLimitMin;
geoGroup.Mechanism.Limitations.OrientationLimitMin.U := 0.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

53.2.4 OrientationLimitMax

Description

The property reads or writes the largest value for the orientation angles in the workspace of the path group. The values are passed and read either individually for the group orientations U, V and W or via a variable of type `Position3D`.

Property	Data type	Meaning
OrientationLimitMax	Position3D	Largest orientation angle set (cardan angles U, V, W).
OrientationLimitMax.U	Double	Largest orientation angle U.
OrientationLimitMax.V	Double	Largest orientation angle V.
OrientationLimitMax.W	Double	Largest orientation angle W.

Syntax

```
OrientationRangeMax := geoGroup.Mechanism.Limitations.OrientationLimitMax;
geoGroup.Mechanism.Limitations.OrientationLimitMax.U := 360.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

53.2.5 OrientationFilter

Description

The property reads or writes the value for the filter setting of the orientation tracking of the path group. Since the orientation is not included in the calculation of the path, it tracks the path movement. Especially with blending movements, moving without a filter would cause the orientation to jump in the blending area. With the filter, however, the orientation is tracked smoothly even in this case.

Property	Data type	Meaning
OrientationFilter	Double	Filter value for orientation tracking. Filter effect in %. 0% = no filter effect; 100% highest filter effect; Default: 70 %; as a rule, this setting should be sufficient.

Syntax

```
OrientationFilter := geoGroup.Mechanism.Limitations.OrientationFilter;
geoGroup.Mechanism.Limitations.OrientationFilter := 50.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 can be determined, for example, by the function of the machine or by the maximum data of the components of the drive system. The setpoint speed of the path is limited to this value, even if the parameterization of the movement would result in a higher speed.

Syntax

```
MaximumSpeed := geoGroup.Mechanism.Limitations.MaxPathVelocity;
geoGroup.Mechanism.Limitations.MaxPathVelocity := 200.5; // mm/s
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 user unit of acceleration. This value can be determined, for example, by the function of the machine or by the maximum data of the components of the drive system. The setpoint acceleration and deceleration of the path are limited to this value, even if the parameterization of the movement would result in a higher acceleration.

Syntax

```
Maximalbeschl := geoGroup.Mechanism.Limitations.MaxPathAcceleration;
geoGroup.Mechanism.Limitations.MaxPathAcceleration := 1300.0; // mm/s
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers 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 user unit of jerk. This value can be determined, for example, by the function of the machine or by the maximum data of the components of the drive system. The setpoint jerk of the path is limited to this value, even if the parameterization of the movement would result in a higher jerk.

Syntax

```
MaximumJerk := geoGroup.Mechanism.Limitations.MaxPathJerk;
geoGroup.Mechanism.Limitations.MaxPathJerk := 100000.0; // mm/s³
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

53.2.9 SingularityDistance1

With serial kinematics, a singularity occurs when the freedom of movement is restricted due to the position of the individual joints. This is the case, for example, when a robot is fully extended. Before a singularity is reached, the speed of a joint may have to become very high in order to follow the setpoint speed of the TCP. It therefore makes sense not only to avoid these singular positions, but also to ensure a certain distance from them. The MCX kernel therefore outputs an error if the position of the TCP comes so close to a singular position that the distance becomes smaller than the limit distance defined via the `SingularityDistance1` property.

Description

Data type	Meaning
Double	Reads or writes the value for the singularity distance that must be maintained to prevent errors due to reaching a singular position [in the user unit of position].

Syntax

```
SingularityLimit := geoGroup.Mechanism.Limitations.SingularityDistance1;
geoGroup.Mechanism.Limitations.SingularityDistance1:= 10.0;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

54 MCGeo.Mechanism.Slope

The **MCGeo.Mechanism.Slope** functionality allows you to read or adjust the calculated properties of the movement of the next travel block of a path group.

A positioning movement consists of the following phases:

1. Acceleration ramp
2. If applicable: travel at constant speed;

   The acceleration ramp can also change directly into the deceleration ramp (triangular mode).
3. Deceleration ramp

In addition, a blending polynomial can be traveled if two path segments are to merge into each other without stopping.

For path groups, the curve of the path speed results from a trapezoidal acceleration profile. The steepness of the ramp in the trapezoid is determined by the specified maximum path jerk. In addition, acceleration and deceleration are the same in the path segment. Both are changed together via the `Acceleration` property.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

54.1 Syntax for MCGeo.Mechanism.Slope

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Mechanism.Slope.<value access>
```

54.2 Properties of MCGeo.Mechanism.Slope

The functionality provides the following properties:

– SlopeState
– IsAccelerating
– IsAtConstantSpeed
– IsDecelerating
– IsStopped
– ActualMaxSpeed
– ActualMaxAcceleration
– SpeedSetpoint
– AccelerationSetpoint
– Speed
– SpeedOverride
– MaxVeloPoly
– Acceleration

54.2.1 SlopeState

Description

Data type	Meaning
MCGeoSlopeStates	Reads which phase of the movement (acceleration, constant speed, deceleration or standstill) the path setpoint of the path group is currently in. This allows you, for example, to activate an action in the machine only after the TCP has left the acceleration phase.
GeoSlopeStates.Accelerating	Group is in the acceleration ramp.
GeoSlopeStates.AtConstandSpeed	Group is moving at constant speed.
GeoSlopeStates.Decelerating	Group is in the deceleration ramp.
GeoSlopeStates.Stopped	Group is at standstill.

Syntax

```
RampStatus := geoGroup.Mechanism.Slope.SlopeState;
```

State management

The property can be read in any state.

54.2.2 IsAccerlerating

Description

Data type	Meaning
Bool	Queries whether the path setpoints are in the acceleration ramp.

Syntax

```
If geoGroup.Mechanism.Slope.IsAccelerating then …
```

State management

The property can be queried in any state.

54.2.3 IsAtConstantSpeed

Description

Data type	Meaning
Bool	Queries whether the path setpoints are in the constant speed phase.

Syntax

```
If geoGroup.Mechanism.Slope.IsAtConstantSpeed then …
```

State management

The property can be queried in any state.

54.2.4 IsDecelerating

Description

Data type	Meaning
Bool	Queries whether the path setpoints are in the deceleration ramp.

Syntax

```
If geoGroup.Mechanism.Slope.IsDecelerating then …
```

State management

The property can be queried in any state.

54.2.5 IsStopped

Description

Data type	Meaning
Bool	Queries whether the path setpoints are in the standstill phase.

Syntax

```
If geoGroup.Mechanism.Slope.IsStopped then …
```

State management

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 setpoint generator of the group. This value differs from the value of the Speed [▶ 303] property, for example, when movements in the preliminary buffer are being processed but the value for the maximum speed has since been changed.

Syntax

```
MaxSpeed := geoGroup.Mechanism.Slope.ActualMaxSpeed;
```

State management

The property can be read in any state.

54.2.7 ActualMaxAcceleration

Description

Data type	Meaning
Double	Reads the value of the maximum path acceleration that is active in the setpoint generator of the group. This value differs from the value of the Acceleration [▶ 304] property, for example, when movements in the preliminary buffer are being processed but the value for the maximum acceleration has since been changed.

Syntax

```
MaxAcceleration := geoGroup.Mechanism.Slope.ActualMaxAcceleration;
```

State management

The property can be read in any state.

54.2.8 SpeedSetpoint

Description

Data type	Meaning
Double	Reads the value of the setpoint path speed currently effective in the setpoint generator of the MCX axis. The sign of the value is always positive.

Syntax

```
ActSpeedSetpoint := geoGroup.Mechanism.Slope.SpeedSetpoint;
```

State management

The property can be read in any state.

54.2.9 AccelerationSetpoint

Description

Data type	Meaning
Double	Reads the value of the setpoint acceleration currently effective in the setpoint generator of the group. The sign of the value indicates acceleration or deceleration.

Syntax

```
ActAccelerationSetpoint := geo.Mechanism.Slope.AccelerationSetpoint;
```

State management

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 for the path setpoint generation of a path movement. This value is normally passed as a parameter in the motion commands. The change of the value only takes effect after the preliminary buffer has been processed.

Syntax

```
ProfileSpeed := geoGroup.Mechanism.Slope.Speed;
geoGroup.Mechanism.Slope.Speed := 300.0;
```

State management

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 for the path setpoint generation for travel on blending polynomials. As a result, the group decelerates to this speed on the preceding travel block if the speed on the travel block was higher. If a higher value than that of the travel block is programmed, it is limited to the travel block speed. The change of the value only takes effect after the preliminary buffer has been processed.

Syntax

```
BlendingSpeed := geoGroup.Mechanism.Slope.MaxVeloPoly;
geoGroup.Mechanism.Slope.MaxVeloPoly := 300.0;
```

State management

The property can be read and written in any state.

54.2.12 SpeedOverride

Description

The speed override changes the setpoint speed in the path setpoint generator by a certain percentage compared with the speed programmed in the motion command.

The maximum possible speed per path movement is determined by the *look-ahead*. The look-ahead calculates the path taking into account the programmed movement and the configured limit values and is therefore also linked to the programmed speed. This can result in a difference depending on whether the group travels, for example, with `Speed = 100mm/s` and `SpeedOveride = 150%` or with `Speed = 150mm/s` and `SpeedOverride = 100%`! To achieve the most uniform speed curve possible, it is therefore advisable to program the highest path speed to be traveled and to use a value ≤ `100 %` for `SpeedOverride`.

The setpoint speed actually calculated in the interpolator is used as the basis for `SpeedOverride`. If, for example, an effective jerk limit results in a lower setpoint speed than the programmed one, the percentage of `SpeedOverride` refers to this lower setpoint speed and not to the programmed speed.

Data type	Meaning
Double	Reads or writes the value for the speed override used for the path setpoint generation of a path movement. This value is normally passed as a parameter in the motion commands. The change of the value takes effect immediately and on all types of path movement, as long as the path movement is not in an acceleration or deceleration ramp. `SpeedOverride` is generally limited to 200 %.

Syntax

```
SpeedOverride := geoGroup.Mechanism.Slope.SpeedOverride;
geoGroup.Mechanism.Slope.SpeedOverride := 50.0;
```

State management

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 for the path setpoint generation of a path movement. This value is normally passed as a parameter in the motion commands. The change of the value only takes effect after the preliminary buffer has been processed.

Syntax

```
ProfileAcceleration := geoGroup.Mechanism.Slope.Acceleration;
geoGroup.Mechanism.Slope.Acceleration := 5500.0;
```

State management

The property can be read and written in any state.

55 MCGeo.Mechanism.Units

The **MCGeo.Mechanism.Units** functionality allows you to read or adjust the units used for the values of positions, speeds, accelerations and jerk.

These units are usually defined during the group definition and can be queried in the STX program using these commands.

NOTICE



Changed positions due to a change of unit during machine operation

It is possible to change the units while the machine is in operation. However, since the positioning command only contains the values without specifying a unit, this changes the meaning of the values.

Example: Changing from millimeters to inches results in a different position in the machine for a value of `25.0` in the positioning command.

- ► Avoid changing the units during operation.
- ► Change the units with care.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

55.1 Syntax for MCGeo.Mechanism.Units

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Mechanism.Units.<function or value access>(optional <parame-
ter 1, parameter 2, parameter n>);
```

55.2 Functions of MCGeo.Mechanism.Units

The functionality provides the following functions:

– SetUnits()

55.2.1 SetUnits()

This function allows you to change all units for positions, speeds, accelerations and jerk for a path movement in one function. The parameters allow you to specify a different meaning for each unit.

Parameters

Parameter	Data type	Meaning
PositionUnit	**MCLinearPositionUnits** (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for the linear position values. Possible units: – Millimeters – Inches – 1/10 millimeters – 1/100 millimeters
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 the linear speed 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 the 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 the linear jerk values. Possible units: – Millimeters per second³ – Meters per second³ – Inches per second³
OrientationUnit	**MCGeoOrientationUnits** (optional) – MCGeoOrientationUnits.RPY_Angles	Unit for the orientation. Possible units: – Cardan angles (see MCGeo.Orientation)

Syntax

```
geoGroup.Mechanism.Units.SetUnits(
    MCLinearPositionUnits.MM,
    MCLinearSpeedUnits.MM_PER_SEC,
    MCLinearAccelerationUnits.MM_PER_S2,
    MCLinearJerkUnits.MM_PER_S3
    MCGeoOrientationUnits.RPY.Angels);
```

State management

The function can be read in any state.

The function can be written in any state. However, it is recommended to change units only in the **Disabled** state.

55.3 Properties of MCGeo.Mechanism.Units

The functionality provides the following properties of the `Mechanism` function:

– PositionUnit
– OrientationUnit
– SpeedUnit
– AccelerationUnit
– JerkUnit

55.3.1 PositionUnit

Description

Reads or writes the unit in which the position values for an axis defined as linear are processed and displayed.

Data type	Meaning
MCLinearPositionUnits (optional) – MCLinearPositionUnits.MM – MCLinearPositionUnits.INCH – MCLinearPositionUnits.MM_10TH – MCLinearPositionUnits.MM_100TH	Unit for the linear position values. Possible units: – Millimeters – Inches – 1/10 millimeters – 1/100 millimeters

Syntax

```
PositionUnits := geoGroup.Mechanism.Units.PositionUnit;
geoGroup.Mechanism.Units.PositionUnit := MCLinearPositionUnits.MM_10TH;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

55.3.2 OrientationUnit

Description

Reads or writes the unit in which the orientation of the path movement is processed and displayed.

Data type	Meaning
MCGeoOrientationUnits (optional) – MCGeoOrientationUnits.RPY_Angles – MCGeoOrientationUnits.TCP_Vector	Unit for the orientation. Possible units: – Cardan angles (see MCGeo.Orientation [▶ 367])

Syntax

```
Ori_Units := geoGroup.Mechanism.Units.OrientationUnit;
geoGroup.Mechanism.Units.OrientationUnit :=
MCLinearOrientationUnits.RPY_Angles;
```

State management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

55.3.3 SpeedUnit

Description

Reads or writes the unit in which the speed values for the 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 the linear speed 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 management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** 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 the linear speed 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 management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** 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 the linear speed 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 management

The property can be read and written in any state.

However, it is recommended to change units only in the **Disabled** state.

56 MCGeo.Members

You use the **MCGeo.Members** functionality primarily to add synchronous axes to a path group or to remove them from the path group. Synchronous axes are axes that are moved in sync with the path movement without being mechanically part of the path movement. A description of the axis types can be found under MCAxis.Mechanism.AxisMode [▶ 114].

For very special cases, you can also add or remove composite axes from the STX program. Since this is only recommended in exceptional cases, the corresponding commands are not described here. Please contact Bucher Automation if you want to use these functions.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

56.1 Syntax for MCGeo.Members

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Members.<function or value access>(optional <parameter>);
```

56.2 Functions of MCGeo.Members

The functionality provides the following functions:

– AddSyncAxis()
– RemoveSyncAxis()

56.2.1 AddSyncAxis()

This function adds a synchronous axis to the path group. It then receives the **Inactive** operating state and is moved via the path group.

Parameters

Parameter	Data type	Meaning
Name of the MCX axis	Pointer to MCAxis	Axis object that is to be moved synchronously with the group.

Syntax

```
geoGroup.Members.AddSyncAxis(axSyncAxis);
```

State management

For the functions to be executed without errors, the path group and the synchronous axis must have the same operating state. This can be either the **Disabled** or the **Enabled** state.

Initial state	Allowed	Final state
Disabled	√	Error: enabled
Enabled	√	Error: enabled

Initial state	Allowed	Final state
Running	X	Error: enabled
Suspended	X	Error: enabled
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

The function can be called in any state of the master and follower axis.

56.2.2 RemoveSyncAxis()

This function removes a synchronous axis from the path group. The removed axis receives the operating state that the path group had before the command was executed. This is either **Enabled** or **Disabled**.

Parameters

Parameter	Data type	Meaning
Name of the MCX axis	Pointer to MCAxis	Axis object that is to be removed from the group.

Syntax

```
geoGroup.Members.RemoveSyncAxis(axSyncAxis);
```

State management

Initial state	Allowed	Final state
Disabled	√	Error: enabled
Enabled	√	Error: enabled
Running	X	Error: enabled
Suspended	X	Error: enabled
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

The function can be called in any state of the master and follower axis.

57 MCGeo.MoveCircle2D

The **MCGeo.MoveCircle2D** functionality allows you to start and parameterize a circular movement of the tool center point (TCP) of an MCX path group in a spatial plane. A linear movement can take place orthogonally to this plane, so that the TCP describes a helical movement.

The circular movement is defined via the center point coordinates, the starting coordinates and the angle of the circular segment. In addition, the length and direction of the movement in the orthogonal direction are specified.

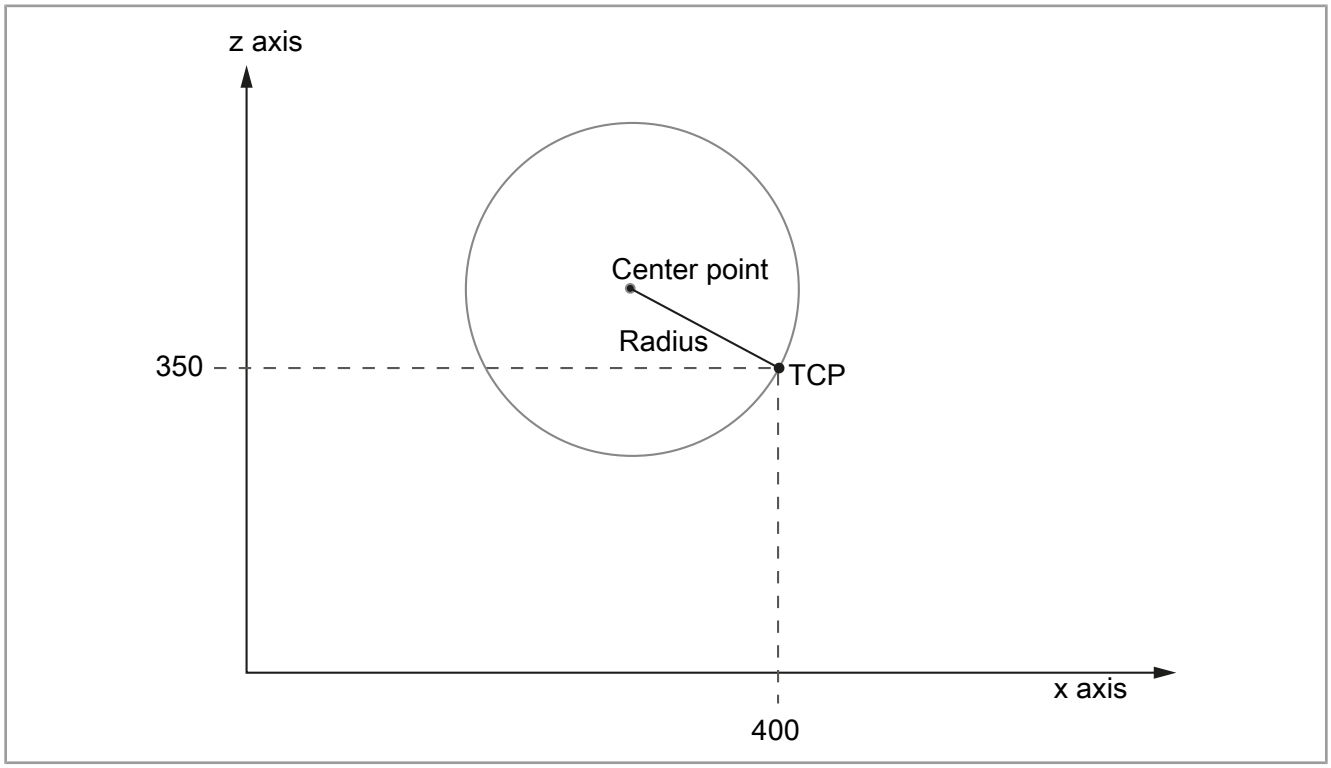

Fig. 11: Principle of the MCGeo.MoveCircle2D functionality

ⓘ Info

Further information

For procedures and sample projects on working with groups, in particular on working with blending, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

57.1 Syntax for MCGeo.MoveCircle2D

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveCircle2D.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

57.2 Functions of MCGeo.MoveCircle2D

The functionality provides the following functions:

– Start()

57.2.1 Start()

The function starts a circular movement of the TCP of a path group in a spatial plane. The circular path is defined by:

– the coordinates of the target point of the last movement;
– the center point coordinates;
– the angle to be covered.

The position of the spatial plane in the orthogonal direction (starting value of the helical movement) is determined by the starting value (target coordinate of the last movement). If the center point coordinate in this direction deviates from the coordinate value of the starting value, it is ignored.

Since a path group works in buffered mode with a preliminary buffer, the first movement is started immediately as a path segment. If further travel commands are written during the movement, these path segments are executed one after the other as soon as the previous path segment has finished. It is possible to have path segments merge into each other by blending without stopping.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.
– The parameters refer to the group object, i.e. they form a property of the group. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode (see PositioningMode [▶ 315]).
BlockEnd	MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started. (block change condition) (see BlockEnd [▶ 316]).
CoordinateSystemPlane	MCGeoCoordinateSystemPlanes (optional)	Coordinate system plane in which the circular segment is to be traveled.
	MCGeoCoordinateSystemPlanes.XY	The circular segment is traveled in the XY plane.
	MCGeoCoordinateSystemPlanes.YZ	The circular segment is traveled in the YZ plane.
	MCGeoCoordinateSystemPlanes.ZX	The circular segment is traveled in the ZX plane.
TravelAngle	Double (optional)	Angle to be covered by the circular segment in [degrees]. The angle is covered in the specified plane in the mathematically positive direction.
TargetPositionHelixAxis	Double (optional)	Target position of the TCP on the coordinate that is orthogonal to the circular plane.
Speed	Double (optional)	Defines the path speed at which the TCP is to be moved as long as the movement is not in the ramps (acceleration, deceleration). The value must be greater than zero.

Parameter	Data type	Meaning
SmoothingDistance	Double (optional)	Blending distance: The value must always be less than half of the shorter of the two path segments that are to merge into each other by blending. The value must be greater than zero, even if the `BlockEnd` condition is selected so that no blending takes place.
Delay	Integer (optional)	Defines the waiting time until the start of the next movement stored in the buffer in [ms]. The waiting time only takes effect 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 path distance for the movement.
CenterPointX CenterPointY CenterPointZ	Double (optional)	Three coordinate parameters for the circle center point of the circular segment to be traveled.
TargetOrientationU TargetOrientationV TargetOrientationW	Double (optional)	Three coordinate parameters for the target orientation of the tool that is to be reached at the end of the path segment.

Syntax

```
geoGroup.MoveCircle2D.Start(MCGeoPositioningModes.Absolute,MCGeoPathBlock-
End.InPosState, MCGeoCoordinateSytem-
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 management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	√		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or the group has been stopped (MCGeo.State [▶ 378]).

```
When geoGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– The **ramp status** indicates which section of the movement the setpoint generator is currently in (MC-Geo.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

The functionality provides the following properties:

– PositioningMode
– BlockEnd
– CoordinateSystemPlane
– TravelAngle
– TargetPositionHelixAxis
– SmoothingDistance
– Delay
– TargetWindow
– CenterPoint
– TargetOrientation

57.3.1 PositioningMode

Description

This property allows you to read the currently valid positioning mode of a path group or to set a new value for this group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the last target position is calculated from the actual position at the time of enabling.

Data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolut	Moves the TCP to the position point that was calculated from the parameters for travel angle, center point coordinates and last target position coordinates. This point is approached relative to the reference position. The travel direction of the path movement results from the sign of the parameter for the travel angle and the helix target position relative to the last target position coordinate in this direction. The orientation axes are controlled so that the new orientation angles are approached by the shortest path. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the negative direction by 50°.
MCGeoPositioningModes.Relative	Specifies both the center point coordinates relative to the starting point and the helix distance relative to the plane at the starting point. The travel direction of the path movement results from the sign of the parameter for the travel angle and the helix target position.
MCGeoPositioningModes.AbsOriNormal	Moves the spatial coordinates of the TCP in the same way as in `Absolute` mode. Controls the orientation axes so that the new orientation angles are approached according to the difference between the last setpoint angle and the target angle. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the positive direction by 310°.

Syntax

```
Geo_Positioniermodus := geoGroup.MoveCircle2D.PositioningMode;
geoGroup.MoveCircle2D.PositioningMode := MCGeoPositioningModes.Relative;
```

State management

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 preliminary buffer of the path group, or to set a new value for this group.

Data type	Meaning
MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment is started when the path segment defined here has reached the setpoint speed `0`.

Data type	Meaning
MCGeoPathBlockEnd.InPos-State_DestWnd	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment is started when the path segment defined here has reached the setpoint speed `0` and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed. In addition, the setpoints of the orientation axes must have reached the target values.
MCGeoPathBlockEnd.Smoothing	The next block change is performed without stopping. Depending on the `SmoothingDistance` parameter, a 5th-degree polynomial is calculated and inserted into the travel buffer as a transition block. This block change condition must not be used for the last path segment in the preliminary buffer.

Syntax

```
Geo_BlockTransition := geoGroup.MoveCircle2D.BlockEnd;
geoGroup.MoveCircle2D.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

State management

The property can be read and written in any state.

57.3.3 CoordinateSystemPlane

Description

This property allows you to read the currently valid value for the coordinate system plane of the 2D circle path segment in the preliminary buffer or of the 2D circle path segment of the path group traveled last. Writing the property sets a new value for this group. It takes effect with the next `MoveCircle2D` command if no other value is written there.

Data type	Meaning
Type MCGeoCoordinateSystem-Planes (optional)	Coordinate system plane in which the circular segment is to be traveled.
MCGeoCoordinateSystemPlanes.XY	The circular segment is traveled in the XY plane.
MCGeoCoordinateSystemPlanes.YZ	The circular segment is traveled in the YZ plane.
MCGeoCoordinateSystemPlanes.ZX	The circular segment is traveled in the ZX plane.

Syntax

```
Segmentebene := geoGroup.MoveCircle2D.CoordinateSystemPlane;
geoGroup.MoveCircle2D.CoordinateSystemPlane :=
MCGeoCoordinateSystemPlanes.ZX;
```

State management

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 circular segment 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 covered with `MoveCircle2D`.

Syntax

```
Segmentwinkel := geoGroup.MoveCircle2D.TravelAngle;
geoGroup.MoveCircle2D.TravelAngle:= 45.2;
```

State management

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 orthogonal to the coordinate system plane of the circular segment.

Syntax

```
Helixposition := geoGroup.MoveCircle2D.TargetPositionHelixAxis;
geoGroup.MoveCircle2D.TargetPositionHelixAxis:= -250.0;
```

State management

The property can be read and written in any state.

57.3.6 SmoothingDistance

Description

Data type	Meaning
Double (optional)	Reads the currently valid value for the blending distance of the path group at the transition from one path segment to the next, or specifies a new value for the path group. The value must not be `0`, even if the `BlockEnd` condition is not `smoothing`.

Syntax

```
Ueberschleifabstand := geoGroup.MoveCircle2D.SmoothingDistance;
geoGroup.MoveCircle2D.SmoothingDistance := 25.6;
```

State management

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 waiting time between two path segments in the preliminary buffer of the path group or specifies a new value for the path group in [ms]. The value only takes effect if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
Wartezeit := geoGroup.MoveCirle2D.Delay;
geoGroup.MoveCircle2D.Delay:= 150;
```

State management

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 this group. This is the path distance from the target point at which the new path segment is started. The value only takes effect if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindowValue := geoGroup. MoveCircle2D.TargetWindow;
geoGroup.MoveCircle2D.TargetWindow := 0.7;
```

State management

The property can be read and written in any state.

57.3.9 CenterPoint

Description

This property allows you to read the current 3D coordinate points of the circle center point of the current path movement of the path group or to specify a new center point for the group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
CenterPoint	Position3D (optional)	Circle center point in 3D direction 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

```
CenterPoint := geoGroup.MoveCircle2D.CenterPoint;
CenterPointX := geoGroup.MoveCircle2D.CenterPoint.X;
geoGroup.MoveCircle2D.CenterPoint := CenterPoint;
```

State management

The property can be read and written in any state.

57.3.10 TargetOrientation

Description

This property allows you to read or specify 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 with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetOrientation	Orientation3D (optional)	Target of the TCP orientation in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value of the TCP orientation in U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value of the TCP orientation in V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
TargetOri3D := geoGroup.MoveCircle2D.TargetOrientation;
TargetU := geoGroup.MoveCircle2D.TargetOrientation.U;
geoGroup.MoveCircle2D.TargetOrientation := NewOrientation;
```

State management

The property can be read and written in any state.

58 MCGeo.MoveCircle3D

The **MCGeo.MoveCircle3D** functionality allows you to start and parameterize a circular movement of the tool center point (TCP) of an MCX path group that lies anywhere in space.

The circular movement is defined via three spatial points that uniquely define the circular segment:

– Starting point (last target point),
– Target point (end point of the circular segment movement),
– An intermediate point on the circular segment.

In the case of 2D kinematics, the Z coordinates of the three points are ignored and the circular segment is traveled in the XY plane.

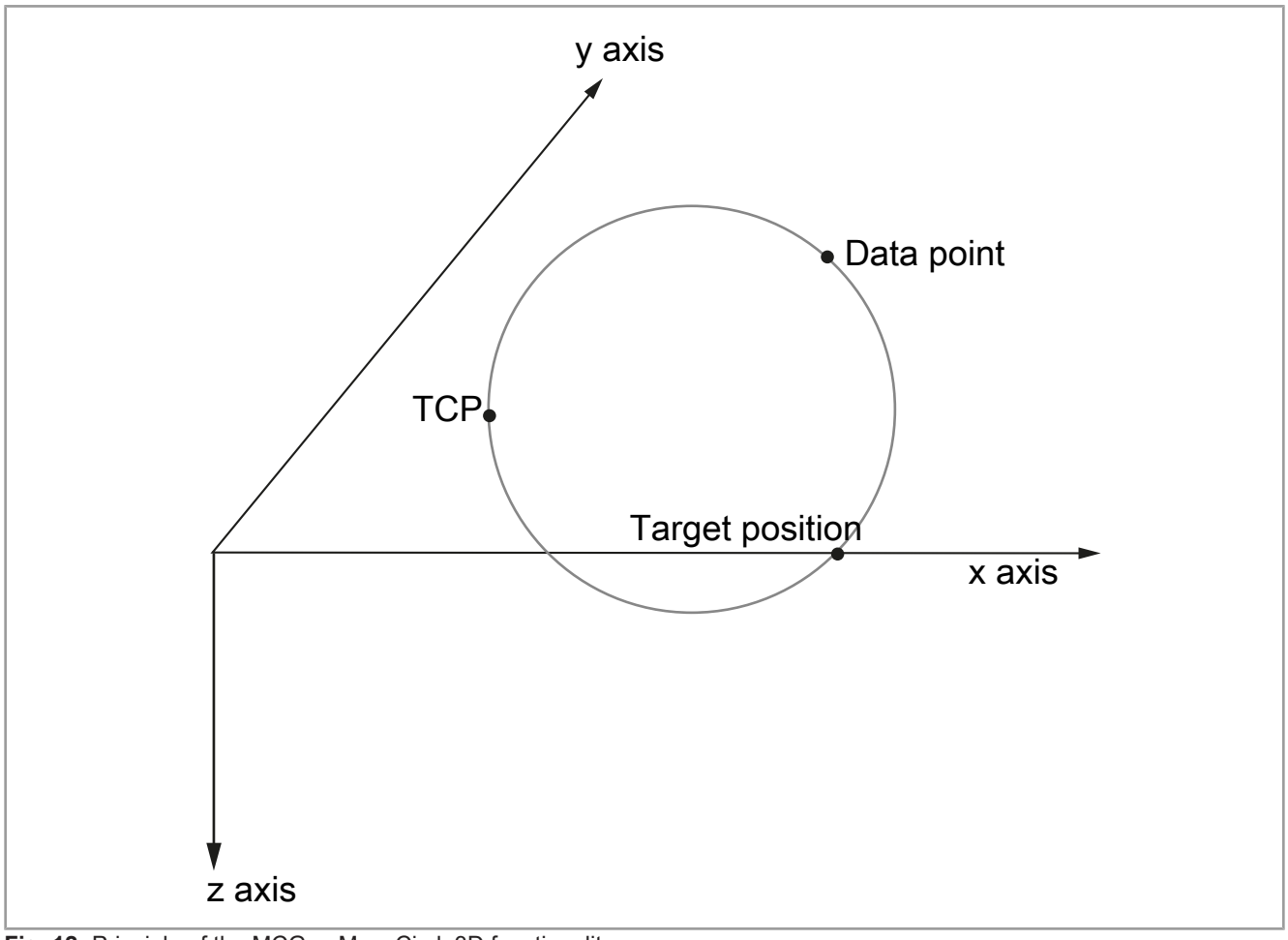

Fig. 12: Principle of the MCGeo.MoveCircle3D functionality

A positioning movement consists of the following phases:

1. Acceleration ramp
2. If applicable: travel at constant speed;

   The acceleration ramp can also change directly into the deceleration ramp (triangular mode).
3. Deceleration ramp

In addition, a blending polynomial can be traveled if two path segments are to merge into each other without stopping.

For path groups, the curve of the path speed results from a trapezoidal acceleration profile. The steepness of the ramp in the trapezoid is determined by the specified maximum path jerk. In addition, acceleration and deceleration are the same in the path segment. Both are changed together via the `Acceleration` property.

Info

Further information

For procedures and sample projects on working with groups, in particular on working with blending, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

58.1 Syntax for MCGeo.MoveCircle3D

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveCircle3D.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

58.2 Functions of MCGeo.MoveCircle3D

The functionality provides the following functions:

– Start()

58.2.1 Start()

The function starts a circular movement of the TCP of a path group 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 circular segment;

– the coordinates of a target point.

The position of the spatial plane in the orthogonal direction (starting value of the helical movement) is determined by the starting value (target coordinate of the last movement). If the center point coordinate in this direction deviates from the coordinate value of the starting value, it is ignored.

Since a path group works in buffered mode with a preliminary buffer, the first movement is started immediately as a path segment. If further travel commands are written during the movement, these path segments are executed one after the other as soon as the previous path segment has finished. It is possible to have path segments merge into each other by blending without stopping.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the group object, i.e. they form a property of the group. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode (see PositioningMode [▶ 324]).
BlockEnd	MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started. (block change condition) (see BlockEnd [▶ 325]).
Speed	Double (optional)	Defines the path speed at which the TCP is to be moved as long as the movement is not in the ramps (acceleration, deceleration). The value must be greater than `0`.

Parameter	Data type	Meaning
SmoothingDistance	Double (optional)	Blending distance: The value must always be less than half of the shorter of the two path segments that are to merge into each other by blending. The value must be greater than `0`, even if the `BlockEnd` condition is selected so that no blending takes place.
Delay	Integer (optional)	Defines the waiting time until the start of the next movement stored in the buffer in [ms]. The waiting time only takes effect 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 path distance for the movement.
IntermediatePointX IntermediatePointY IntermediatePointZ	Double (optional)	Three coordinate parameters for the intermediate point on the circular segment to be traveled.
TargetPosiitionX TargetPosiitionY TargetPosiitionZ	Type Double [optional]	Three coordinate parameters for the path target point of the circular segment to be traveled.
TargetOrientationU TargetOrientationV TargetOrientationW	Double (optional)	Three coordinate parameters for the target orientation of the tool that 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 management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	√		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or the group has been stopped (MCGeo.State [▶ 378]).

```
When geoGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– The **ramp status** indicates which section of the movement 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

The functionality provides the following properties:

– PositioningMode
– BlockEnd
– SmoothingDistance
– Delay
– TargetWindow
– IntermediatePoint
– TargetPosition
– TargetOrientation

58.3.1 PositioningMode

Description

This property allows you to read the currently valid positioning mode of a path group or to set a new value for this group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the last target position is calculated from the actual position at the time of enabling.

Data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolut	Moves the TCP to the position point that was calculated from the parameters for travel angle, center point coordinates and last target position coordinates. This point is approached relative to the reference position. The travel direction of the path movement results from the sign of the parameter for the travel angle and the helix target position relative to the last target position coordinate in this direction. The orientation axes are controlled so that the new orientation angles are approached by the shortest path. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the negative direction by 50°.
MCGeoPositioningModes.Relative	Specifies both the center point coordinates relative to the starting point and the helix distance relative to the plane at the starting point. The travel direction of the path movement results from the sign of the parameter for the travel angle and the helix target position.
MCGeoPositioningModes.AbsOriNormal	Moves the spatial coordinates of the TCP in the same way as in `Absolute` mode. Controls the orientation axes so that the new orientation angles are approached according to the difference between the last setpoint angle and the target angle. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the positive direction by 310°.

Syntax

```
Geo_Positioniermodus := geoGroup.MoveCircle3D.PositioningMode;
geoGroup.MoveCircle3D.PositioningMode := MCGeoPositioningModes.Relative;
```

State management

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 preliminary buffer of the path group, or to set a new value for this group.

Data type	Meaning
MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment is started when the path segment defined here has reached the setpoint speed `0`.

Data type	Meaning
MCGeoPathBlockEnd.InPos-State_DestWnd	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment is started when the path segment defined here has reached the setpoint speed `0` and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed. In addition, the setpoints of the orientation axes must have reached the target values.
MCGeoPathBlockEnd.Smoothing	The next block change is performed without stopping. Depending on the `SmoothingDistance` parameter, a 5th-degree polynomial is calculated and inserted into the travel buffer as a transition block. This block change condition must not be used for the last path segment in the preliminary buffer.

Syntax

```
Geo_BlockTransition := geoGroup.MoveCircle3D.BlockEnd;
geoGroup.MoveCircle3D.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

State management

The property can be read and written in any state.

58.3.3 SmoothingDistance

Description

Data type	Meaning
Double (optional)	Reads the currently valid value for the blending distance of the path group at the transition from one path segment to the next, or specifies a new value for the path group. The value must not be `0`, even if the `BlockEnd` condition is not `smoothing`.

Syntax

```
Ueberschleifabstand := geoGroup.MoveCircle3D.SmoothingDistance;
geoGroup.MoveCircle3D.SmoothingDistance := 25.6;
```

State management

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 waiting time between two path segments in the preliminary buffer of the path group or specifies a new value for the path group in [ms]. The value only takes effect if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
Wartezeit := geoGroup.MoveCirle3D.Delay;
geoGroup.MoveCircle3D.Delay:= 150;
```

State management

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 this group. This is the path distance from the target point at which the new path segment is started. The value only takes effect if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindowValue := geoGroup. MoveCircle3D.TargetWindow;
geoGroup.MoveCircle3D.TargetWindow := 0.7;
```

State management

The property can be read and written in any state.

58.3.6 IntermediatePoint

Description

This property allows you to read the current 3D coordinate points of the intermediate point of the current path movement of the path group or to specify a new intermediate point for the group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
IntermediatePoint	Position3D (optional)	Intermediate point on the circular segment in 3D direction coordinates.
IntermediatePoint.X	Position3D.X (Double)	X coordinate of the intermediate point.
IntermediatePoint.Y	Position3D.Y (Double)	Y coordinate of the intermediate point.
IntermediatePoint.Z	Position3D.Z (Double)	Z coordinate of the intermediate point.

Syntax

```
Zwischenpunkt := geoGroup.MoveCircle2D.IntermediatePoint;
ZwischenpunktX := geoGroup.MoveCircle2D.IntermediatePoint.X;
geoGroup.MoveCircle2D.IntermediatePoint := Zwischenpunkt;
```

State management

The property can be read and written in any state.

58.3.7 TargetPosition

Description

This property allows you to read the current 3D coordinate points of the target of 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 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
TargetPosition	Position3D (optional)	Target of the TCP position in 3D direction coordinates.
TargetPosition.X	Position3D.X (Double)	Target value of the TCP position in X direction.
TargetPosition.Y	Position3D.Y (Double)	Target value of the TCP position in Y direction.
TargetPosition.Z	Position3D.Z (Double)	Target value of the TCP position in Z direction.

Syntax

```
Target3D := geoGroup.MoveCircle3D.TargetPosition;
TargetX := geoGroup.MoveCircle3D.TargetPosition.X;
geoGroup.MoveCircle3D.TargetPosition := NewTargetPoint;
```

State management

The property can be read and written in any state.

58.3.8 TargetOrientation

Description

This property allows you to read or specify the current 3D coordinate values of the target of the current path movement of the path group.

The coordinate values are passed in a structure with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetOrientation	Orientation3D (optional)	Target of the TCP orientation in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value of the TCP orientation in U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value of the TCP orientation in V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
TargetOri3D := geoGroup.MoveCircle3D.TargetOrientation;
TargetU := geoGroup.MoveCircle3D.TargetOrientation.U;
geoGroup.MoveCircle3D.TargetOrientation := NewOrientation;
```

State management

The property can be read and written in any state.

59 MCGeo.MoveLinear

The **MCGeo.MoveLinear** functionality allows you to start and parameterize a linear path movement of the tool center point (TCP) of an MCX path group.

A positioning movement consists of the following phases:

1. Acceleration ramp
2. If applicable: travel at constant speed;

   The acceleration ramp can also change directly into the deceleration ramp (triangular mode).
3. Deceleration ramp

In addition, a blending polynomial can be traveled if two path segments are to merge into each other without stopping.

For path groups, the curve of the path speed results from a trapezoidal acceleration profile. The steepness of the ramp in the trapezoid is determined by the specified maximum path jerk. In addition, acceleration and deceleration are the same in the path segment. Both are changed together via the `Acceleration` property.

ⓘ Info

Further information

For procedures and sample projects on working with groups, in particular on working with blending, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

59.1 Syntax for MCGeo.MoveLinear

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveLinear.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

59.2 Functions of MCGeo.MoveLinear

The functionality provides the following functions:

– Start()

59.2.1 Start()

The function starts a linear movement of the TCP of a path group.

Since a path group works in buffered mode with a preliminary buffer, the first movement is started immediately as a path segment. If further travel commands are written during the movement, these path segments are executed one after the other as soon as the previous path segment has finished. It is possible to have path segments merge into each other by blending without stopping.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the group object, i.e. they form a property of the group. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode (see PositioningMode [▶ 332]).
BlockEnd	MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started. (block change condition) (see BlockEnd [▶ 333]).
Speed	Double (optional)	Defines the path speed at which the TCP is to be moved as long as the movement is not in the ramps (acceleration, deceleration). The value must be greater than `0`.
SmoothingDistance	Double (optional)	Blending distance: The value must always be less than half of the shorter of the two path segments that are to merge into each other by blending. The value must be greater than `0`, even if the `BlockEnd` condition is selected so that no blending takes place.
Delay	Integer (optional)	Defines the waiting time until the start of the next movement stored in the buffer in [ms]. The waiting time only takes effect 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 path distance for the movement.
TargetPosiitionX TargetPosiitionY TargetPosiitionZ	Type Double [optional]	Three coordinate parameters for the target position in space at which the TCP is to stand at the end of the path movement.
TargetOrientationU TargetOrientationV TargetOrientationW	Double (optional)	Three coordinate parameters for the target orientation of the tool that 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 management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	√		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or the group has been stopped (MCGeo.State [▶ 378]).

```
When geoGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– The **ramp status** indicates which section of the movement 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

The functionality provides the following properties:

– PositioningMode
– BlockEnd
– SmoothingDistance
– Delay
– TargetWindow
– TargetCoordinates
– TargetPosition
– TargetOrientation

59.3.1 PositioningMode

Description

This property allows you to read the currently valid positioning mode of a path group or to set a new value for this group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the last target position is calculated from the actual position at the time of enabling.

Data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolut	Moves the TCP to the position point that was calculated from the parameters for `TargetPosition`. This point is approached relative to the reference position. The travel direction of the path movement results from the last and the new target position. The orientation axes are controlled so that the new orientation angles are approached by the shortest path. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the negative direction by 50°.
MCGeoPositioningModes.Relative	Moves the TCP further by the distance passed in the `TargetPosition` parameter. The target point is calculated from the coordinates of the last target point plus the value of the distances in `TargetPosition`. The distance coordinates can be specified with a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	Moves the spatial coordinates of the TCP in the same way as in `Absolute` mode. Controls the orientation axes so that the new orientation angles are approached according to the difference between the last setpoint angle and the target angle. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the positive direction by 310°.

Syntax

```
Geo_Positioniermodus := geoGroup.MoveLinear.PositioningMode;
geoGroup.MoveLinear.PositioningMode := MCGeoPositioningModes.Relative;
```

State management

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 preliminary buffer of the path group, or to set a new value for this group.

Data type	Meaning
MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment is started when the path segment defined here has reached the setpoint speed `0`.

Data type	Meaning
MCGeoPathBlockEnd.InPos-State_DestWnd	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed.
MCGeoPathBlockEnd.InPos-State_SyncOri	The next path segment is started when the path segment defined here has reached the setpoint speed `0` and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed. In addition, the setpoints of the orientation axes must have reached the target values.
MCGeoPathBlockEnd.Smoothing	The next block change is performed without stopping. Depending on the `SmoothingDistance` parameter, a 5th-degree polynomial is calculated and inserted into the travel buffer as a transition block. This block change condition must not be used for the last path segment in the preliminary buffer.

Syntax

```
Geo_BlockTransition := geoGroup.MoveLinear.BlockEnd;
geoGroup.MoveLinear.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

State management

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 blending distance of the path group at the transition from one path segment to the next, or specifies a new value for the path group. The value must not be `0`, even if the `BlockEnd` condition is not `smoothing`.

Syntax

```
Ueberschleifabstand := geoGroup.MoveLinear.SmoothingDistance;
geoGroup.MoveLinear.SmoothingDistance := 25.6;
```

State management

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 waiting time between two path segments in the preliminary buffer of the path group or specifies a new value for the path group in [ms]. The value only takes effect if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
Wartezeit := geoGroup.MoveLinear.Delay;
geoGroup.MoveLinear.Delay:= 150;
```

State management

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 this group. This is the path distance from the target point at which the new path segment is started. The value only takes effect if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindowValue := geoGroup. MoveLinear.TargetWindow;
geoGroup.MoveLinear.TargetWindow := 0.7;
```

State management

The property can be read and written in any state.

59.3.6 TargetCoordinates

Description

This property allows you to read the current 6D coordinate points of the target of 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 3 direction coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetCoordinates	Coordinate6D (optional)	Target of the TCP position in 6D spatial coordinates.
TargetCoordinates.X	Coordinate6D.X (Double)	Target value of the TCP position in X direction.
TargetCoordinates.Y	Coordinate6D.Y (Double)	Target value of the TCP position in Y direction.

TargetOrientation	Data type	Meaning
TargetCoordinates.Z	Coordinate6D.Z (Double)	Target value of the TCP position in Z direction.
TargetCoordinates.U	Coordinate6D.U (Double)	Target value of the TCP orientation in U direction.
TargetCoordinates.V	Coordinate6D.V (Double)	Target value of the TCP orientation in V direction.
TargetCoordinates.W	Coordinate6D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
Target6D := geoGroup.MoveLinear.TargetCoordinates;
TargetY := geoGroup.MoveLinear.TargetCoordinates.Y;
geoGroup.MoveLinear.TargetCoordinates := NewTargetPoint;
```

State management

The property can be read and written in any state.

59.3.7 TargetPosition

Description

This property allows you to read the current 3D coordinate points of the target of 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 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
TargetPosition	Position3D (optional)	Target of the TCP position in 3D direction coordinates.
TargetPosition.X	Position3D.X (Double)	Target value of the TCP position in X direction.
TargetPosition.Y	Position3D.Y (Double)	Target value of the TCP position in Y direction.
TargetPosition.Z	Position3D.Z (Double)	Target value of the TCP position in Z direction.

Syntax

```
Target3D := geoGroup.MoveLinear.TargetPosition;
TargetX := geoGroup.MoveLinear.TargetPosition.X;
geoGroup.MoveLinear.TargetPosition := NewTargetPoint;
```

State management

The property can be read and written in any state.

59.3.8 TargetOrientation

Description

This property allows you to read or specify the current 3D coordinate values of the target of the current path movement of the path group.

The coordinate values are passed in a structure with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetOrientation	Orientation3D (optional)	Target of the TCP orientation in 3D orientation coordinates.

TargetOrientation	Data type	Meaning
TargetOrientation.U	Orientation3D.U (Double)	Target value of the TCP orientation in U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value of the TCP orientation in V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
TargetOri3D := geoGroup.MoveLinear.TargetOrientation;
TargetU := geoGroup.MoveLinear.TargetOrientation.U;
geoGroup.MoveLinear.TargetOrientation := NewOrientation;
```

State management

The property can be read and written in any state.

60 MCGeo.MoveOrientation

The **MCGeo.MoveOrientation** functionality allows you to start and/or parameterize a 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. Blending of this movement is not possible.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

60.1 Syntax for MCGeo.MoveOrientation

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveOrientation.<function or value access>(optional <parame-
ter 1, parameter 2, parameter n>);
```

60.2 Functions of MCGeo.MoveOrientation

The functionality provides the following functions:

– Start()

60.2.1 Start()

The function starts an orientation movement of the TCP of a path group. Since a path group works in buffered mode with a preliminary buffer, the first movement is started immediately as a path segment. If further travel commands are written during the movement, these path segments are executed one after the other as soon as the previous path segment has finished. It is possible to have path segments merge into each other by blending without stopping.

Parameters

Parameter	Data type	Meaning
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode (see PositioningMode [▶ 340]).
BlockEnd	MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
Speed	Double (optional)	Defines the speed of the movement in [°/s].
Acceleration	Double (optional)	Defines the acceleration of the movement in [°/s²].
Jerk	Double (optional)	Defines the jerk of the movement in [°/s³].

Parameter	Data type	Meaning
Delay	Integer (optional)	Defines the waiting time until the start of the next movement stored in the buffer in [ms]. The waiting time only takes effect if the positioning mode `InPosState_DestWnd_Delay` has been selected.
TargetWindow	Double (optional)	Defines the width of the target window relative to the path distance in [°].
TargetOrientationU TargetOrientationV TargetOrientationW	Double (optional)	Three coordinate parameters for the target orientation of the tool that 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 management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	√		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or the group has been stopped (MCGeo.State [▶ 378]).

```
When geoGroup.State.IsEnabled Continue; //the axis is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– The **ramp status** indicates which section of the movement 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

The functionality provides the following properties:

– PositioningMode
– BlockEnd
– Speed
– Acceleration
– Jerk
– Delay
– TargetWindow
– TargetOrientation

60.3.1 PositioningMode

Description

This property allows you to read the currently valid positioning mode of a path group or to set a new value for this group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the last target position is calculated from the actual position at the time of enabling.

Data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolut	Moves the orientation of the TCP to the values passed in the TargetOrientation parameter. It is approached relative to the reference position. The orientation axes are controlled so that the new orientation angles are approached by the shortest path. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the negative direction by 50°.
MCGeoPositioningModes.Relative	Moves the orientation of the TCP further by the values passed in the `TargetOrientation` parameter. The target point is calculated from the coordinates of the last target point plus the value of the distances in `TargetOrientation`. The values can be specified with a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	Moves the orientation axes so that the new orientation angles are approached according to the difference between the last setpoint angle and the target angle. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the positive direction by 310°.

Syntax

```
Geo_Positioniermodus := geoGroup.MoveOrientation.PositioningMode;
geoGroup.MoveOrientation.PositioningMode := MCGeoPositioningModes.Relative;
```

State management

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 preliminary buffer of the path group, or to set a new value for this group.

Data type	Meaning
MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment is started when the axis has reached the setpoint speed zero.
MCGeoPathBlockEnd.InPos-State_DestWnd	The next path segment is started when the setpoint position of the axis with the longest travel distance has reached the target position up to the target window distance.
MCGeoPathBlockEnd.InPos-State_DestWnd_Delay	The next path segment is started when the setpoint position of the axis with the longest travel distance has reached the target position and the dwell time has elapsed.

Syntax

```
Geo_BlockTransition := geoGroup.MoveOrientation.BlockEnd;
geoGroup.MoveOrientation.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

State management

The property can be read and written in any state.

60.3.3 Speed

Description

Data type	Meaning
Double (optional)	Reads the currently valid value for the orientation speed or specifies a new value in [°/s]. This parameter defines the orientation speed at which the TCP is to be moved as long as the movement is not in the ramps (acceleration, deceleration). The value must be greater than zero.

Syntax

```
OriSpeed := geoGroup.MoveOrientation.Speed;
geoGroup.MoveOrientation.Speed:= 15.4;
```

State management

The property can be read and written in any state.

60.3.4 Acceleration

Description

Data type	Meaning
Double (optional)	Reads the currently valid value for the orientation acceleration or specifies a new value in [°/s²]. This parameter defines the orientation acceleration with which the TCP is to be moved as long as the movement is in the ramps (acceleration, deceleration). The value must be greater than zero.

Syntax

```
OriRampe := geoGroup.MoveOrientation.Acceleration;
geoGroup.MoveOrientation.Acceleration:= 120.1;
```

State management

The property can be read and written in any state.

60.3.5 Jerk

Description

Data type	Meaning
Double (optional)	Reads the currently valid value for the orientation jerk or specifies a new value in [°/s³]. This parameter defines the orientation jerk with which the TCP is to be moved as long as the movement is in the ramps (acceleration, deceleration). The jerk defines the steepness of the ramps. The value must be greater than zero.

Syntax

```
OriJerk := geoGroup.MoveOrientation.Jerk;
geoGroup.MoveOrientation.Jerk:= 1514.4;
```

State management

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 waiting time between two path segments in the preliminary buffer of the path group, or specifies a new value for the path group in [ms]. The value only takes effect if `InPosState_DestWnd_Delay` is used as the `BlockEnd` condition.

Syntax

```
Wartezeit := geoGroup.MoveOrientation.Delay;
geoGroup.MoveOrientation.Delay:= 150;
```

State management

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 setpoint angle distance from the target point of the orientation at which the new path segment is started, or writes a new value for this group. The value only takes effect if `InPosState_DestWnd` or `InPosState_DestWnd_Delay` is used as the `BlockEnd` condition.

Syntax

```
TargetWindowValue := geoGroup.MoveOrientation.TargetWindow;
geoGroup.MoveOrientation.TargetWindow := 0.7;
```

State management

The property can be read and written in any state.

60.3.8 TargetOrientation

Description

This property allows you to read or specify 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 with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetOrientation	Orientation3D (optional)	Target of the TCP orientation in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value of the TCP orientation in U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value of the TCP orientation in V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
TargetOri3D := geoGroup.MoveOrientation.TargetOrientation;
TargetU := geoGroup.MovePto.TargetOrientation.U;
geoGroup.MoveOrientation.TargetOrientation := NewOrientation;
```

State management

The property can be read and written in any state.

61 MCGeo.MovePtp

The **MCGeo.MovePtp** functionality allows you to start and/or parameterize a point-to-point movement of the tool center point (TCP) of an MCX path group in space.

Unlike the other `MCGeo.Move` commands, `MovePtp` does not describe a prescribed path. The axes of the path group are controlled to their target so that all axes arrive at the same time. The axis with the longest travel time determines the travel speed. The other axes are moved more slowly according to their travel distance. Depending on the design of the kinematics and the start and target position in space, this results in the path curve. Blending into a Ptp movement and out of a Ptp movement is not possible.

The movement with MovePtp is often the fastest movement of the TCP in space and is used if the path curve does not matter for the function and if it is certain that the movement space is free of obstacles.

ⓘ Info

Further information

For procedures and sample projects on working with groups, in particular on working with blending, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

61.1 Syntax for MCGeo.MovePtp

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MovePtp.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

61.2 Functions of MCGeo.MovePtp

The functionality provides the following functions:

– Start()

61.2.1 Start()

The function starts an orientation movement of the TCP of a path group. Since a path group works in buffered mode with a preliminary buffer, the first movement is started immediately as a path segment. If further travel commands are written during the movement, these path segments are executed one after the other as soon as the previous path segment has finished. It is possible to have path segments merge into each other by blending without stopping.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the group.

– The parameters refer to the group object, i.e. they form a property of the group. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode (see PositioningMode [▶ 346]).

Parameter	Data type	Meaning
BlockEnd	MCGeoMovePtpBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
Speed	Double (optional)	Defines the speed of the movement. It is specified as a percentage of the maximum speed of the axis with the longest travel distance. The other axes are moved more slowly according to the length of their travel distance.
Delay	Integer (optional)	Defines the waiting time until the start of the next movement stored in the buffer in [ms]. The waiting 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 at which the TCP is to stand after the path movement has been executed.
TargetOrientationU TargetOrientationV TargetOrientationW	Double (optional)	Three coordinate parameters for the target orientation of the tool that is to be reached at the end of the path segment.

Syntax

```
geoGroup.MovePtp.Start(MCGeoPositioningModes.Absolute,MCGeoPathBlockEnd.In-
PosState,85.0,0.0,0.1,100.0,50.0,150.0,0.0,0.0,0.0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	√		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or the group has been stopped (MCGeo.State [▶ 378]).

```
When geoGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– The **ramp status** indicates which section of the movement 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

The functionality provides the following properties:

– PositioningMode
– BlockEnd
– Delay
– TargetWindow
– TargetPosition
– TargetOrientation
– TargetCoordinates

61.3.1 PositioningMode

Description

This property allows you to read the currently valid positioning mode of a path group or to set a new value for this group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the last target position is calculated from the actual position at the time of enabling.

Data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolut	Moves the orientation of the TCP to the values passed in the `TargetOrientation` parameter. It is approached relative to the reference position. The travel direction of the positioning results from the last and the new target position. The orientation axes are controlled so that the new orientation angles are approached by the shortest path. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the negative direction by 50°.

Data type	Meaning
MCGeoPositioningModes.Relative	Moves the orientation of the TCP further by the values passed in the `TargetOrientation` parameter. The target point is calculated from the coordinates of the last target point plus the value of the distances in `TargetOrientation`. The values can be specified with a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	Moves the orientation axes so that the new orientation angles are approached according to the difference between the last setpoint angle and the target angle. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the positive direction by 310°.

Syntax

```
Geo_Positioniermodus := geoGroup.MovePtp.PositioningMode;
geoGroup.MovePtp.PositioningMode := MCGeoPositioningModes.Relative;
```

State management

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 preliminary buffer of the path group, or to set a new value for this group.

Data type	Meaning
MCGeoMovePtpBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
MCGeoMovePtpBlockEnd.InPosState	The next path segment is started when the axis has reached the setpoint speed zero.
MCGeoMovePtpBlockEnd.InPosState_DestWnd	The next path segment is started when the setpoint position of the axis with the longest travel distance has reached the target position up to the target window distance.
MCGeoMovePtpBlockEnd.InPosState_DestWnd_Delay	The next path segment is started when the setpoint position of the axis with the longest travel distance has reached the target position and the dwell time has elapsed.

Syntax

```
Geo_BlockTransition := geoGroup.MovePtp.BlockEnd;
geoGroup.MovePtp.BlockEnd := MCGeoMovePtpBlockEnd.InPosState_DestWnd;
```

State management

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 waiting time between two path segments in the preliminary buffer of the path group, or specifies a new value for the path group in [ms]. The value only takes effect if `InPosState_DestWnd_Delay` is used as the `BlockEnd` condition.

Syntax

```
Wartezeit := geoGroup.MovePtp.Delay;
geoGroup.MovePtp.Delay:= 150;
```

State management

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 setpoint angle distance from the target point of the orientation at which the new path segment is started, or writes a new value for this group. The value only takes effect if `InPosState_DestWnd` or `InPosState_DestWnd_Delay` is used as the `BlockEnd` condition.

Syntax

```
TargetWindowValue := geoGroup.MovePtp.TargetWindow;
geoGroup.MovePtp.TargetWindow := 0.7;
```

State management

The property can be read and written in any state.

61.3.5 TargetPosition

Description

This property allows you to read the current 3D coordinate points of the target of 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 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
TargetPosition	Position3D (optional)	Target of the TCP position with 3D direction coordinates.
TargetPosition.X	Position3D.X (Double)	Target value of the TCP position in X direction.
TargetPosition.Y	Position3D.Y (Double)	Target value of the TCP position in Y direction.
TargetPosition.Z	Position3D.Z (Double)	Target value of the TCP position in Z direction.

Syntax

```
Target3D := geoGroup.MovePtp.TargetPosition;
TargetX := geoGroup.MovePtp.TargetPosition.X;
geoGroup.MovePtp.TargetPosition := NewTargetPoint;
```

State management

The property can be read and written in any state.

61.3.6 TargetOrientation

Description

This property allows you to read or specify 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 with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetOrientation	Orientation3D (optional)	Target of the TCP orientation in 3D orientation coordinates.
TargetOrientation.U	Orientation3D.U (Double)	Target value of the TCP orientation in U direction.
TargetOrientation.V	Orientation3D.V (Double)	Target value of the TCP orientation in V direction.
TargetOrientation.W	Orientation3D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
TargetOri3D := geoGroup.MovePtp.TargetOrientation;
TargetU := geoGroup.MovePto.TargetOrientation.U;
geoGroup.MovePtp.TargetOrientation := NewOrientation;
```

State management

The property can be read and written in any state.

61.3.7 TargetCoordinates

Description

This property allows you to read the current 6D coordinate points of the target of 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 3 direction coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

TargetOrientation	Data type	Meaning
TargetCoordinates	Coordinate6D (optional)	Target of the TCP position in 6D spatial coordinates.
TargetCoordinates.X	Coordinate6D.X (Double)	Target value of the TCP position in X direction.
TargetCoordinates.Y	Coordinate6D.Y (Double)	Target value of the TCP position in Y direction.
TargetCoordinates.Z	Coordinate6D.Z (Double)	Target value of the TCP position in Z direction.

TargetOrientation	Data type	Meaning
TargetCoordinates.U	Coordinate6D.U (Double)	Target value of the TCP orientation in U direction.
TargetCoordinates.V	Coordinate6D.V (Double)	Target value of the TCP orientation in V direction.
TargetCoordinates.W	Coordinate6D.W (Double)	Target value of the TCP orientation in W direction.

Syntax

```
Target6D := geoGroup.MovePtp.TargetCoordinates;
TargetY := geoGroup.MovePtp.TargetCoordinates.Y;
geoGroup.MovePtp.TargetCoordinates := NewTargetPoint;
```

State management

The property can be read and written in any state.

62 MCGeo.MoveSpline

The **MCGeo.MoveSpline** functionality allows you to start or parameterize a spline-shaped path movement of the tool center point (TCP) of an MCX path group in space. With this type of interpolation, the path curve is approximated by cubic splines at up to 6 data points. Cubic splines are third-degree polynomials that are continuous up to the second derivative. The starting point (last target point of the previous path command) is included in the polynomial calculation as the first point.

Further explanations can be found in the JetSym help on the topic Spline.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 50 – 52, 54 – 57** and the documents *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

62.1 Syntax for MCGeo.MoveSpline

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveSpline.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

62.2 Functions of MCGeo.MoveCircle2D

The functionality provides the following functions:

– Start()

62.2.1 Start()

The function starts a spline movement of the TCP of a path group. Since a path group works in buffered mode with a preliminary buffer, the first movement is started immediately as a path segment. If further travel commands are written during the movement, these path segments are executed one after the other as soon as the previous path segment has finished. It is possible to have path segments merge into each other by blending without stopping.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the group.

– The parameters refer to the group object, i.e. they form a property of the group. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
Nodes	Integer (optional)	Number of data points to be used for the spline movement (at least **2,** at most 6).
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode.
BlockEnd	MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition). See BlockEnd [▶ 356]

Parameter	Data type	Meaning
StartCondition	Type MCGeoMoveSplineBlockTransitions (optional)	Defines the start condition of the spline movement (see StartCondition [▶ 356]).
TargetCondition	Type MCGeoMoveSplineBlockTransitions (optional)	Defines the target condition of the spline movement (see TargetCondition [▶ 357]).
Speed	Double (optional)	Defines the path speed at which the TCP is to be positioned as long as the movement is not in the ramps (acceleration, deceleration). The value must be greater than zero.
SmoothingDistance	Double (optional)	Defines the blending distance: The value must always be less than half of the shorter of the two path segments that are to merge into each other by blending. The value must be greater than zero, even if the `BlockEnd` condition is selected so that no blending takes place.
Delay	**Integer** (optional)	Defines the waiting time until the next movement in the buffer is started in [ms]. The waiting time 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 path distance for the movement.
VirtualStartPositionX VirtualStartPositionY VirtualStartPositionZ	Double (optional)	Specifies an imaginary starting position that is used for the tangent calculation for the start condition `ExplicitTangent`. **NOTICE! Note that the virtual starting position is usually not reached by the path. A tangent to the path at the first data point is traveled.**
VirtualTargetPositionX VirtualTargetPositionY VirtualTargetPositionZ	Double (optional)	Specifies an imaginary target position that is used for the tangent calculation for the target condition `ExplicitTangent`. **NOTICE! Note that the virtual target position is usually not reached by the path. A tangent to the path at the last data point is traveled.**
Node0PositionX Node0PositionY Node0PositionZ	Double (optional)	Defines three coordinate parameters for the first data point in space that are used to calculate the path movement.
Node0OrientationU Node0OrientationV Node0OrientationW	Double (optional)	Defines three coordinate parameters for the orientation of the tool at the first data point that are used to calculate the path.
….		Coordinates and orientations for data points 1 … 4 analogous to `Node0` and `Node5`.

Parameter	Data type	Meaning
Node5PositionX Node5PositionY Node5PositionZ	Double (optional)	Defines three coordinate parameters for the sixth data point in space that 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 data point that are used to calculate the path.

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
, // Deceleration - 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 management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	√		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the path movement:

– The **state of the group** indicates whether the setpoint generator is generating a movement or the group has been stopped (MCGeo.State [▶ 378]).

```
When geoGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
```

```
When geoGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– The **ramp status** indicates which section of the movement the setpoint generator is currently in (MC-Geo.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

The functionality provides the following properties:

– NodeCount
– PositioningMode
– BlockEnd
– StartCondition
– TargetCondition
– SmoothingDistance
– 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 data points of a spline movement of the path group or specifies a new value for the path group. Smallest value = 2, largest value = 6.

Syntax

```
DataPointCount := geoGroup.MoveSpline.NodeCount;
geoGroup.MoveSpline.NodeCount:= 6;
```

State management

The property can be read and written in any state.

62.3.2 PositioningMode

Description

This property allows you to read the currently valid positioning mode of a path group or to set a new value for this group.

ⓘ Info

Determining the target position after the Enable() command

After enabling (`Enable()`), the last target position is calculated from the actual position at the time of enabling.

Data type	Meaning
MCGeoPositioningModes (optional)	Mode for the type of path movement.
MCGeoPositioningModes.Absolut	Moves the TCP to the position point passed in the parameters for the last `NodeTargetPosition`. This point is approached relative to the reference position. The travel direction of the positioning results from the last and the new target position. The orientation axes are controlled so that the new orientation angles are approached by the shortest path. The spatial coordinates of the TCP move in the same way as in `AbsOriNormal` mode. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the negative direction by 50°.
MCGeoPositioningModes.Relative	Moves the TCP further by the distance passed in the parameters for the `NodeTargetPosition` for the individual data points. The target point is calculated from the coordinates of the last target point plus the value of the distances in `NodeTargetPosition`. The distance coordinates can be specified with a positive or negative sign.
MCGeoPositioningModes.AbsOriNormal	The spatial coordinates of the TCP move in the same way as in `Absolute` mode. Moves the orientation axes so that the new orientation angles are approached according to the difference between the last setpoint angle and the target angle. **Example**: With a last target orientation position of 40° and a new target orientation position of 350°, the orientation axis moves in the positive direction by 310°.

Syntax

```
Geo_Positioniermodus := geoGroup.MoveSpline.PositioningMode;
geoGroup.MoveSpline.PositioningMode := MCGeoPositioningModes.Relative;
```

State management

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 preliminary buffer of the path group, or to set a new value for this group.

Data type	Meaning
MCGeoPathBlockEnd (optional)	Completion condition of a path segment in the preliminary buffer. It defines when the next path segment in the preliminary buffer is started (block change condition).
MCGeoPathBlockEnd.InPosState	The next path segment is started when the axis has reached the setpoint speed zero.
MCGeoPathBlockEnd.InPosState_DestWnd	The next path segment is started when the setpoint position of the axis with the longest travel distance has reached the target position up to the target window distance.
MCGeoPathBlockEnd.InPosState_DestWnd_Delay	The next path segment is started when the setpoint position of the axis with the longest travel distance has reached the target position and the dwell time has elapsed.
MCGeoPathBlockEnd.InPosState_SyncOri	The next path segment is started when the path segment defined here has reached the setpoint speed `0` and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPosState_DestWnd_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position up to the target window distance and the setpoints of the orientation axes have reached the target values.
MCGeoPathBlockEnd.InPosState_DestWnd_Delay_SyncOri	The next path segment is started when the setpoint position of the path has reached the target position and the dwell time has elapsed. In addition, the setpoints of the orientation axes must have reached the target values.
MCGeoPathBlockEnd.Smoothing	The next block change is performed without stopping. Depending on the `SmoothingDistance` parameter, a 5th-degree polynomial is calculated and inserted into the travel buffer as a transition block.

Syntax

```
Geo_BlockTransition := geoGroup.MoveSpline.BlockEnd;
geoGroup.MoveSpline.BlockEnd := MCGeoPathBlockEnd.InPosState_DestWnd;
```

State management

The property can be read and written in any state.

62.3.4 StartCondition

Description

This property allows you to read the currently valid start condition for the motion control of the spline movement or to specify a new value.

Data type	Meaning
MCGeoMoveSplineBlockTransitions (optional)	Start condition for the spline movement.

Data type	Meaning
MCGeoMoveSplineBlockTransitions.ExplicitTangent	The *explicit tangent specification* option requires a virtual starting position. This position is used together with the first data point to calculate the tangent direction with which the first data point is approached. Note that the virtual starting position is usually not reached by the path. A tangent to the path at the first data point is traveled.
MCGeoMoveSplineBlockTransitions.AutomaticMode	In *automatic* mode, the transition from the previous block is calculated automatically. Normally, a parabolic path through the first three points is used. If there is no previous block, the TCP is accelerated with the value set in the parameter for the path acceleration.
MCGeoMoveSplineBlockTransitions.Tangential	The *tangential to previous block* option has the same behavior as `ExplicitTangent`, except that the virtual starting position is defined by the last target position of the previous block and is approached in the previous block.
MCGeoMoveSplineBlockTransitions.NaturalMode	In *natural mode*, the transition from the previous block is calculated in such a way that the TCP is not accelerated during the transition. This means that the transition takes place at a constant path speed. If there is no previous block, the TCP is accelerated with the value set in the parameter for the path acceleration.

Syntax

```
Geo_Startverahlten := geoGroup.MoveSpline.StartCondition;
geoGroup.MoveSpline.StartCondition :=
MCGeoMoveSplineBlockTransitions.AutomaticMode;
```

State management

The property can be read and written in any state.

62.3.5 TargetCondition

Description

This property allows you to read the currently valid target condition for the motion control of the spline movement or to specify a new value.

Data type	Meaning
MCGeoMoveSplineBlockTransitions (optional)	Target condition for the spline movement.
MCGeoMoveSplineBlockTransitions.ExplicitTangent	The *explicit tangent specification* option requires a virtual starting position. This position is used together with the first data point to calculate the tangent direction with which the first data point is approached. Note that the virtual starting position is usually not reached by the path. A tangent to the path at the first data point is traveled.
MCGeoMoveSplineBlockTransitions.AutomaticMode	In *automatic* mode, the transition from the previous block is calculated automatically. Normally, a parabolic path through the first three points is used. If there is no previous block, the TCP is accelerated with the value set in the parameter for the path acceleration.
MCGeoMoveSplineBlockTransitions.Tangential	The *tangential* to previous block option has the same behavior as `ExplicitTangent`, except that the virtual starting position is defined by the last target position of the previous block and is approached in the previous block.

Data type	Meaning
MCGeoMoveSplineBlockTransitions.NaturalMode	In *natural mode*, the transition from the previous block is calculated in such a way that the TCP is not accelerated during the transition. This means that the transition takes place at a constant path speed. If there is no previous block, the TCP is accelerated with the value set in the parameter for the path acceleration.

Syntax

```
Geo_TargetBehavior := geoGroup.MoveSpline.TargetCondition;
geoGroup.MoveSpline.TargetCondition :=
MCGeoMoveSplineBlockTransitions.Tangential;
```

State management

The property can be read and written in any state.

62.3.6 SmoothingDistance

Description

Data type	Meaning
Double (optional)	Reads the currently valid value for the blending distance of the path group at the transition from one path segment to the next, or specifies a new value for the path group. The value must not be `0`, even if the `BlockEnd` condition is not `smoothing`.

Syntax

```
Ueberschleifabstand := geoGroup.MoveSpline.SmoothingDistance;
geoGroup.MoveLinear.SmoothingDistance := 25.6;
```

State management

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 waiting time between two path segments in the preliminary buffer of the path group or specifies a new value for the path group in [ms]. The value only takes effect if `InPosState_DestWnd_Delay` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
Wartezeit := geoGroup.MoveSpline?.Delay;
geoGroup.MoveLinear.Delay:= 150;
```

State management

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 this group. This is the path distance from the target point at which the new path segment is started. The value only takes effect if `InPosState_DestWnd`, `InPosState_DestWnd_Delay`, `InPosState_DestWnd_SyncOri` or `InPosState_DestWnd_Delay_SyncOri` is used as the `BlockEnd` condition.

Syntax

```
TargetWindowValue := geoGroup. MoveSpline.TargetWindow;
geoGroup.MoveLinear.TargetWindow := 0.7;
```

State management

The property can be read and written in any state.

62.3.9 VirtualStartPosition[x]

Description

This property allows you to read the current 3D coordinate points of the virtual starting position of the current spline movement of the path group or to specify a new virtual starting position for the spline movement of the group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
VirtualStartPosition	Position3D (optional)	Position for the virtual starting position of the TCP in 3D direction coordinates.
VirtualStartPosition.X	Position3D.X (Double)	Position for the virtual starting position of the TCP in X direction.
VirtualStartPosition.Y	Position3D.Y (Double)	Position for the virtual starting position of the TCP in Y direction.
VirtualStartPosition.Z	Position3D.Z (Double)	Position for the virtual starting position of the TCP in Z direction.

Syntax

```
VirtStartPosition_3D := geoGroup.MoveSpline.VirtualStartPosition;
VirtStartPositionX := geoGroup.MoveSpline.VirtualStartPosition.X;
Geo.MoveSpline. VirtualStartPosition:= VirtStartPos_3D;
```

State management

The property can be read and written in any state.

62.3.10 VirtualTargetPosition[x]

Description

This property allows you to read the current 3D coordinate points of the virtual target position of the current spline movement of the path group or to specify a new virtual target position for the spline movement of the group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
VirtualTargetPosition	Position3D (optional)	Position for the virtual target position of the TCP in 3D direction coordinates.
VirtualTargetPosition.X	Position3D.X (Double)	Position for the virtual target position of the TCP in X direction.
VirtualTargetPosition.Y	Position3D.Y (Double)	Position for the virtual target position of the TCP in Y direction.
VirtualTargetPosition.Z	Position3D.Z (Double)	Position for the virtual target position of the TCP in Z direction.

Syntax

```
VirtTargetPosition_3D := geoGroup.MoveSpline.VirtualTargetPosition;
VirtTargetPositionX := geoGroup.MoveSpline.VirtualTargetPosition.X;
Geo.MoveSpline. VirtualTargetPosition:= VirtTargetPos_3D;
```

State management

The property can be read and written in any state.

62.3.11 NodePosition[n]

Description

This property allows you to read the current 3D coordinate points of the data points of the current spline movement of the path group or to specify a new target point for the group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
NodePosition[n]	Position3D (optional)	Position for the data points of the TCP in 3D direction coordinates.
NodePosition[n].X	Position3D.X (Double)	Position for the data point of the TCP in X direction.
NodePosition[n].Y	Position3D.Y (Double)	Position for the data point of the TCP in Y direction.
NodePosition[n].Z	Position3D.Z (Double)	Position for the data point of the TCP in Z direction.

Where *[n]* is the number of the data point.

Syntax

```
DataPoint3_3D := geoGroup.MoveSpline.NodePosition[3];
DataPoint3X := geoGroup.MoveSpline.NodePosition[3].X;
Geo.MoveSpline.NodePosition[2]:= Pos2_3D;
```

State management

The property can be read and written in any state.

62.3.12 NodeOrientation[n]

Description

This property allows you to read the current 3D coordinate points of the data points of the current spline movement of the path group or to specify a new target point for the group.

The coordinate values are passed in a structure with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
NodePosition[n]	Position3D (optional)	Target of the orientation for the data points of the TCP in 3D orientation coordinates.
NodePosition[n].U	Position3D.U (Double)	Orientation for the data point of the TCP in U direction.
NodePosition[n].V	Position3D.V (Double)	Orientation for the data point of the TCP in V direction.
NodePosition[n].W	Position3D.W (Double)	Orientation for the data point of the TCP in W direction.

Syntax

```
DataPointOri2_3D := geoGroup.MoveSpline.NodeOrientation[2];
DataPoint0U := geoGroup.MoveSpline.NodeOrientation[0].U;
geoGroup.MoveSpline.NodeOrientation[3] := Orientation3_3D;
```

State management

The property can be read and written in any state.

62.3.13 NodeCoordinates[n]

Description

This property allows you to read the current 6D coordinate points of the data points of the current spline movement of the path group or to specify a new data point for the group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z) and 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
NodeCoordinates[n]	Coordinate6D (optional)	Data point coordinates of the TCP in 6D spatial coordinates.
NodeCoordinates[n].X	Coordinate6D.X (Double)	Position for the data point of the TCP in X direction.
NodeCoordinates[n].Y	Coordinate6D.Y (Double)	Position for the data point of the TCP in Y direction.
NodeCoordinates[n].Z	Coordinate6D.Z (Double)	Position for the data point of the TCP in Z direction.
NodeCoordinates[n].U	Coordinate6D.U (Double)	Orientation of the data points of the TCP in U direction.
NodeCoordinates[n].V	Coordinate6D.V (Double)	Orientation of the data points of the TCP in V direction.
NodeCoordinates[n].W	Coordinate6D.W (Double)	Orientation of the data points of the TCP in W direction.

Where *[n]* is the number of the data point.

Syntax

```
DataPoint2_6D := geoGroup.MoveSpline.NodeCoordinates[2];
DataPoint0Y := geoGroup.MoveSpline.NodeCoordinates[0].Y;
geoGroup.MoveSpline.NodeCoordinates[4]:= DataPoint4_6D;
```

State management

The property can be read and written in any state.

63 MCGeo.MoveSyncAxis

The **MCGeo.MoveSyncAxis** functionality allows you to query information about synchronous axes of a path group and to assign motion parameters to the synchronous axes so that they move with the path group in the desired way.

ⓘ Info

Further information

For procedures and sample projects on working with synchronous axes in path groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group* and *Path group – synchronous axis* . These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

63.1 Syntax for MCGeo.MoveSyncAxis

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveSyncAxis.<function or value access>(optional <parameter>);
```

63.2 Functions of MCGeo.MoveSyncAxis

The functionality provides the following functions:

- Start()
- GetTargetPosition()
- SetTargetPosition()
- GetPositioningMode()
- SetPositioningMode()

63.2.1 Start()

The function adds a new movement for the synchronous axis to the path buffer. The movement is started synchronously with the next path movements of the path group. The synchronous movement is distributed over the path movements in such a way that the target position (`TargetPosition` parameter) is reached when the path group stops with the `BlockEnd` parameter `InPosState`. If several path segments that are joined without stopping (`BlockEnd` parameter `Smoothing`) are written to the path buffer after the `Start()` command for the synchronous axis, the movement of the synchronous axis is executed across these path segments and only reaches the target position when the next path segment stops with the `Blockend` parameter `InPosState`.

Parameters

Parameter	Data type	Meaning
Name of the MC axis	Pointer to MCAxis	Axis object that is to be moved synchronously with the group. This axis must have been added to the path group beforehand with AddSyncAxis() [▶ 310].
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode for the synchronous movement.
	MCGeoPositioning-Modes.Absolute	With the next path movement, the synchronous axis is moved to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the last and the new target position.
	MCGeoPositioning-Modes.Relative	With the next path movement, the synchronous axis is moved further by the distance passed in the `TargetPosition` parameter. The target position is calculated from the last target position plus the value in `TargetPosition`. The travel direction depends on the sign of the `TargetPosition` parameter.
TargetPosition	Double (optional)	Defines the target position for the synchronous movement.

Syntax

```
geoGroup.MoveSyncAxis.Start(AxSync, MCGeoPositioningModes.Absolute,300.0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	√	Enabled
Running	√	Enabled
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

63.2.2 GetTargetPosition()

Parameters

Parameter	Data type	Meaning
Name of the MC axis	Pointer to MCAxis	Reads the value last passed for the target position of the synchronous axis. This axis must have been added to the path group beforehand with AddSyncAxis() [▶ 310].

Syntax

```
TargetPos := geoGroup.MoveSyncAxis.GetTargetPosition(AxSync);
```

State management

The function can be read in any state.

63.2.3 SetTargetPosition()

Parameters

Parameter	Data type	Meaning
Name of the MC axis	Pointer to MCAxis	Passes a new value for the target position of the synchronous axis. This axis must have been added to the path group beforehand with AddSyncAxis() [▶ 310]. This value is only stored in the axis object. It only takes effect with a Start() [▶ 363] command in which no target position is passed.

Syntax

```
geoGroup.MoveSyncAxis.SetTargetPosition(AxSync,400.0);
```

State management

The following table applies to writing the function:

63.2.4 GetPositioningMode()

Parameters

Parameter	Data type	Meaning
Name of the MC axis	Pointer to MCAxis	Reads the value last passed for the positioning mode of the synchronous axis. The axis must have been added to the path group beforehand with AddSyncAxis() [▶ 310].

Syntax

```
PosMod := geoGroup.MoveSyncAxis.GetPositioningMode(AxSync);
```

State management

The function can be read in any state.

63.2.5 SetPositioningMode()

Parameters

Parameter	Data type	Meaning
Name of the MC axis	Pointer to MCAxis	Axis object that is to be moved synchronously with the group. This axis must have been added to the path group beforehand with AddSyncAxis() [▶ 310].
PositioningMode	MCGeoPositioningModes (optional)	Defines the positioning mode for the synchronous movement.
	MCGeoPositioning-Modes.Absolute	With the next path movement, the synchronous axis is moved to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the last and the new target position.
	MCGeoPositioning-Modes.Relative	With the next path movement, the synchronous axis is moved further by the distance passed in the `TargetPosition` parameter. The target position is calculated from the last target position plus the value in `TargetPosition`. The travel direction depends on the sign of the `TargetPosition` parameter.

Syntax

```
geoGroup.MoveSyncAxis.SetPositioningMode(AxSync,MCGeoPositioningModes.Absolute);
```

State management

The following table applies to writing the function:

64 MCGeo.Orientation

For kinematics that have one or more orientation axes, the **MCGeo.Orientation** functionality allows you to query the current values for the 3D orientation sorted by coordinates.

The orientation of the tool center point (TCP) is described by the so-called **cardan angles**, in the rotation axis sequence ***z – y‘ – x“***, with respect to the base coordinate system.

Determining the cardan angles

To derive the coordinate system of the TCP ***(x‘“, y‘“, z‘“)*** from the base coordinate system (x, y, z), the following scheme is applied:

1. First, the fixed coordinate system is rotated by the angle U about the ***z*** axis. This results in the coordinate system ***(x‘, y‘, z‘)***.
2. This is now rotated by the angle V about the ***y‘*** axis, resulting in the coordinate system ***(x“, y“, z“)***.
3. Finally, this coordinate system is rotated by the angle W about the ***x“*** axis. This results in the coordinate system of the TCP ***(x‘“, y‘“, z‘“)***.

Meaning of the cardan angles

– The angle U (also yaw angle) describes the alignment of the TCP with respect to the ***x-y*** plane of the base coordinate system.

– The angle V (also pitch angle) describes the alignment of the TCP with respect to the ***y-z*** plane of the base coordinate system.

– The angle W (also roll angle) describes the rotation of the TCP about the ***x*** axis of the TCP coordinate system.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

64.1 Syntax for MCGeo.Orientation

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Orientation.<value access>;
```

64.2 Properties of MCGeo.Orientation

The functionality provides the following properties:

– Actual

– Setpoint

– Target

64.2.1 Actual

Description

This property allows you to read the current 3D coordinate points of the path group.

The coordinate values are passed in a structure with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. 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 U direction.
Actual.V	Orientation3D.V (Double)	Current value of the TCP orientation in V direction.
Actual.W	Orientation3D.W (Double)	Current value of the TCP orientation in W direction.

Syntax

```
ActualOri3D := geoGroup.Orientation.Actual;
ActualOriU := geoGroup.Orientation.Actual.U;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

64.2.2 Setpoint

Description

This property allows you to read the current 3D coordinate points of the setpoint generator of the path group.

The coordinate values are passed in a structure with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Setpoint	Orientation3D	Setpoint of the TCP orientation in 3D orientation coordinates.
Setpoint.U	Orientation3D.U (Double)	Setpoint of the TCP orientation in U direction.
Setpoint.V	Orientation3D.V (Double)	Setpoint of the TCP orientation in V direction.
Setpoint.W	Orientation3D.W (Double)	Setpoint of the TCP orientation in W direction.

Syntax

```
SetpointOri3D := geoGroup.Orientation.Setpoint;
SetpointU := geoGroup.Orientation.Setpoint.U;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

64.2.3 Target

Description

This property allows you 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 with 3 orientation coordinates (U, V, W). Only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Target	Orientation3D	Target of the TCP orientation in 3D orientation coordinates.
Target.U	Orientation3D.U (Double)	Target of the TCP orientation in U direction.
Target.V	Orientation3D.V (Double)	Target of the TCP orientation in V direction.
Target.W	Orientation3D.W (Double)	Target of the TCP orientation in W direction.

Syntax

```
TargetOri3D := geoGroup.Orientation.Target;
TargetU := geoGroup.Orientation.Target.U;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

65 MCGeo.Position

The **MCGeo.Position** functionality allows you to query the current values for the 3D path movement sorted by coordinates.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

65.1 Syntax for MCGeo.Position

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Position.<value access>;
```

65.2 Properties of MCGeo.Position

The functionality provides the following properties:

– Actual
– Setpoint
– Target

65.2.1 Actual

Description

This property allows you to read the current 3D coordinate points of the path group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Actual	Position3D	Actual value of the TCP position in 3D direction coordinates.
Actual.X	Position3D.X (Double)	Current value of the TCP position in X direction
Actual.Y	Position3D.Y (Double)	Current value of the TCP position in Y direction
Actual.Z	Position3D.Z (Double)	Current value of the TCP position in Z direction

Syntax

```
Actual3D := geoGroup.Position.Actual;
ActualX := geoGroup.Position.Actual.X;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

65.2.2 Setpoint

Description

This property allows you to read the current 3D coordinate points of the setpoint generator of the path group.

The coordinate values are passed in a structure with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Setpoint	Position3D	Setpoint of the TCP position in 3D spatial coordinates.
Setpoint.X	Position3D.X (Double)	Setpoint of the TCP position in X direction.
Setpoint.Y	Position3D.Y (Double)	Setpoint of the TCP position in Y direction.
Setpoint.Z	Position3D.Z (Double)	Setpoint of the TCP position in Z direction.

Syntax

```
Setpoint3D := geoGroup.Position.Setpoint;
SetpointX := geoGroup.Position.Setpoint.X;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

65.2.3 Target

Description

This property allows you 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 with 3 direction coordinates (X, Y, Z). Always only the values that are relevant for the kinematics of the path group are passed. The other values remain zero.

Property	Data type	Meaning
Target	Position3D	Target of the TCP position in 3D spatial coordinates.
Target.X	Position3D.X (Double)	Target of the TCP position in X direction.
Target.Y	Position3D.Y (Double)	Target of the TCP position in Y direction.
Target.Z	Position3D.Z (Double)	Target of the TCP position in Z direction.

Syntax

```
Target3D := geoGroup.Position.Target;
TargetX := geoGroup.Position.Target.X;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

66 MCGeo.Power

The **MCGeo.Power** functionality 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 at the same time.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

66.1 Syntax for MCGeo.Power

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Power.<function>(optional <parameter>;
```

66.2 Functions of MCGeo.Power

The functionality provides the following functions:

– Enable()
– Disable()
– Quickstop()

66.2.1 Enable()

The function activates the controllers of the servo amplifiers of all composite axes and enables the voltage to the motor. A path group can only be enabled if all composite axes are ready for operation.

Parameters

The function has no parameters.

Syntax

```
geoGroup.Power.Enable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Enabled
Enabled	–	Enabled
Running	–	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, query in the program whether the new state has been reached before calling further commands for the group.

```
When geoGroup.State.IsEnabled Continue;
```

66.2.2 Disable()

The function disables the servo amplifiers of all composite axes for torque generation (pulse inhibit). The motors are de-energized immediately and coast to a stop if no brake becomes active.

Parameters

Parameter	Data type	Meaning
DisableModes	MCPowerDisableMode (optional)	Mode for executing the `Disable()` function.
	MCPowerDisableMode.Forced (default)	The pulse inhibit always takes effect immediately. This mode is also active if no parameter is specified.
	MCPowerDisableMode.StateCheck	The pulse inhibit only takes effect if the group is in the **Enabled** operating state. In all other states, an operating error is output.

Syntax

```
geoGroup.Power.Disable();
geoGroup.Power.Disable(MCPowerDisableModes.Forced);
geoGroup.Power.Disable(MCPowerDisableModes.StateCheck);
```

State management

The following table applies to writing the function:

The following table applies to calling without a parameter or with the `MCPowerDisableModes.Forced` parameter:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	√	Disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

The following table applies to calling with the `MCPowerDisableModes.StateCheck` parameter:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled

Initial state	Allowed	Final state
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, before calling further commands for the group, query in the program whether the new state has been reached, as follows:

```
When 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()

The function stops all composite axes with the quick stop deceleration and disables the servo amplifiers once standstill has been reached.

NOTICE



No position control of the composite axes during quick stop

During the quick stop deceleration, there is no position control and therefore no coordinated movement of the composite axes according to the path movement. Each composite axis comes to a standstill at a different time depending on its current speed and the setting for the quick stop deceleration.

► Only use the command in emergencies!

Parameters

The function has no parameters.

Syntax

```
geoGroup.Power.QuickStop();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	√	Disabled

Initial state	Allowed	Final state
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, before calling further commands for the group, query in the program whether the new state has been reached, as follows:

```
When geoGroup.State.IsDisabled Continue;
```

66.3 Example task MCGeo.Power

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample1 Autorun
    //Enable the composite axes
    If geoGroup.State.IsDisabled Then
        geoGroup.Power.Enable();
    End_if;
    When geoGroup.State.IsEnabled Continue;

    //a process with the group can be placed here, or other tasks move the
group
    //…

    // Disable the group
    If geoGroup.State.IsRunning Then    //if the group is still moving, stop
it with a suitable deceleration

        geoGroup.MoveHalt.Start();
    When geoGroup.Mechanism.Slope.IsStopped Continue;

    End_if;
    geoGroup.Power.Disable(); // Disable the group
    When geoGroup.State.IsDisabled Continue;

End_Task;
```

67 MCGeo.Settings

The **MCGeo.Settings** functionality allows you to adjust or save all parameters of an MCX path group together.

Do not adjust parameters while saving is in progress

Saving the parameters to a file is a process that is executed in the background, i.e. it extends over several MCX cycles. If parameter values are changed during saving, it is unpredictable whether the old or the new value is written to the file.

How parameter management works

Parameters saved in the motion setup of JetSym are stored in a text file on the PC and, on request, also in the file system of the control unit (subfolder Motion). The file name of the parameter file on the control unit is *MCGeo_X.ini*, where *X* stands for the number of the path group ID. The path group ID is displayed on the Properties page in the motion setup of the MCX group. In addition, the tooltip in the hardware tree shows the path group ID when the mouse pointer is on the group.)

When the control unit boots, the MCX kernel reads the data of the *MCGeo_X.ini* file and uses it for the group concerned. If such a file is present on the control unit, no further parameterization is required. If there is no corresponding file for the group, the parameters are loaded with default values.

Normally, no adjustment to the file from the STX program is required.

In very few exceptional cases, however, it can make sense to read in the parameter file on the control unit again or to save the parameters. When saving, however, bear in mind that the parameter file saved in the JetSym project may then no longer match the parameters effective on the machine.

67.1 Syntax for MCGeo.Settings

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Settings.<function>(optional <parameter>;
```

67.2 Functions of MCGeo.Settings

The functionality provides the following functions:

– SaveToFile()
– LoadFromFile()

67.2.1 SaveToFile()

Parameters

Parameter	Data type	Meaning
Suffix	Integer (optional)	Activates saving of the parameters of the MCX group to a file in the subfolder Motion in the file system of the control unit. Depending on the parameter passed, different files are created or overwritten: – Parameter `0` or no parameter: The parameter file *MCGeo_X.ini* for this group is created or overwritten with the current values. – Parameter not equal to `0`: The parameter file *MCGeo_X_TmpY.ini* is created or overwritten with the current values. *Y* is equal to the `Suffix` parameter.

Syntax

```
geoGroup.Settings.SaveToFile();    // overwrites (or creates) the file MC-
Geo_1.ini (if geoGroup has Geo ID 1)
geoGroup.Settings.SaveToFile(3);    // creates the file MCGeo_1_Tmp3.ini (if
geoGroup has Geo ID 1)
```

State management

The function can be called in any state.

67.2.2 LoadFromFile()

Parameters

Parameter	Data type	Description
Suffix	Integer (optional)	Activates the adoption of the parameters of the MCX group from a file in the subfolder Motion in the file system of the control unit. Depending on the parameter passed, different files are read. – Parameter `0` or no parameter: The data from the file *MCGeo_X.ini* is read and used for the group concerned. – Parameter not equal to `0`: The data from the file *MCGeo_TmpY.ini* is read and used for the group concerned. *Y* is equal to the `Suffix` parameter.

Syntax

```
geoGroup.Settings.LoadFromFile();    //Reads the file MCGeo_1.ini (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 management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

68 MCGeo.State

The **MCGeo.State** functionality allows you to query various information about a path group, such as operating mode, motion type, etc.

Basic and further information

In this context, also observe the following information:

– For state transitions, section MCGeo.State.Transitions [▶ 389];

– For a detailed description of the structure of the MCX core, the JetSym online help (JetSym Motion and MotionControl > JetSym MotionControl > MotionControl basics > Operating concept > Overview: Operating concept)

– For basic information, Motion Control Application Note No. 54 (MCX motion control basics).

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

Operating mode

The operating mode (`OperatingMode`) indicates how the path group basically works. There are 3 operating modes:

Operating mode	Meaning
`Inactive` (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
`Normal` (normal)	The group can be used. Travel commands are executed in buffered mode (see MCGeo.Buffer [▶ 257]), i.e. all motion commands are added to a preliminary buffer and processed one after the other.
`Error` (error)	An operating error is pending. The operating state is retained, but no new functions and parameters can be passed.

Tab. 6: Operating modes of MCX path groups

Operating state

At any time, a group is in one, and only one, clearly defined operating state (`OperatingState`). If, for example, the group is in the **Running** operating state, the **Enabled** operating state is not active, although of course all composite axes have pulse enable.

In a particular state, always only a subset of all function calls is permitted. The state model for the axis is as follows:

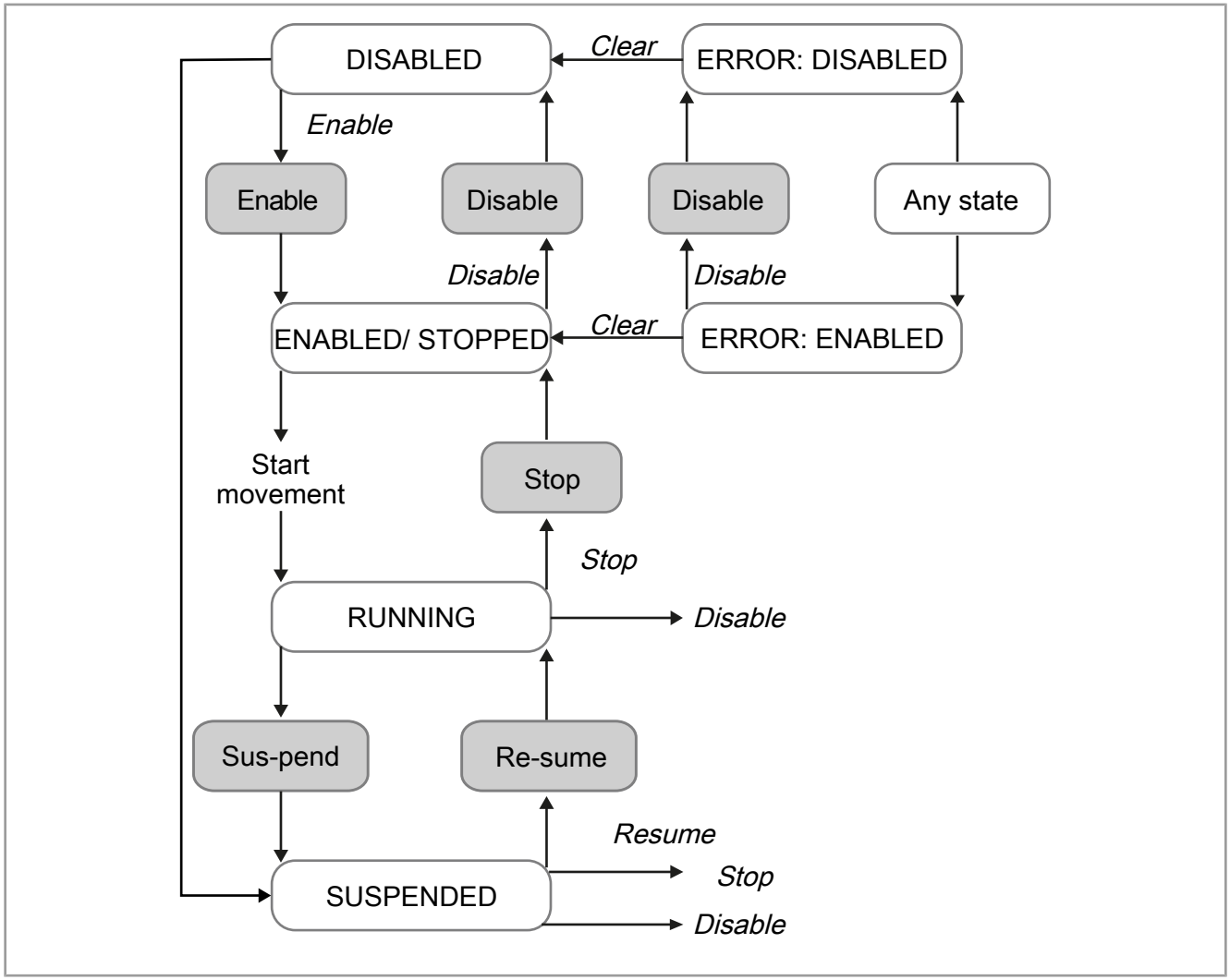

Fig. 13: State model for MCX groups

Operating state	Meaning
Inactive (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
Disabled (disabled)	– The controllers of the servo amplifiers of all composite axes are open (no pulse enable). – No torque is generated.
Enabled (enabled)	– The controllers of all composite axes are closed (pulse enable). – The setpoint interpolator is at standstill.
Running (running)	At least one axis is moving, i.e. the setpoint interpolator passes motion setpoints on to the servo amplifier.
Suspended (suspended)	– Corresponds to the **Enabled** operating state, with the exception that the target position is retained. A movement suspended with `MCGeo.State.Transitions.Suspend()` can be continued with the command `MCGeo.State.Transitions.Resume()` (state transition resume).
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control (error level 4).
DisabledError (Error: disabled)	An error has occurred. After the error response, the control of all axes is disabled (error levels 5 and 6).

Tab. 7: Operating states of path groups

Motion type

The motion type (`TypeOfMotion`) indicates which type of movement the setpoint generator is currently generating for the path group.

Querying the motion type is useful, for example, in a monitoring task in which the process is not known. Based on the motion type active before switching off, it is possible to control the restart correctly.

Operating state	Meaning
No movement	This motion type is active until the first movement of the group.
Straight line (`MoveLinear`)	The path group is currently executing, or last executed, a linear movement.
Circular movement 2D (`MoveCircular2D`)	The path group is currently executing, or last executed, a 2D circular movement.
Circular movement 3D (`MoveCircular3D`)	The path group is currently executing, or last executed, a 3D circular movement.
Spline (`MoveSpline`)	The path group is currently executing, or last executed, a spline movement.
Polynomial (`MovePolynomial`)	The path group is executing a blending movement.
Point-to-point (`MovePtp`)	The path group is currently executing, or last executed, a point-to-point movement.
Orientation (`MoveOrientation`)	The path group is currently executing, or last executed, an orientation movement.
Coupled movement (`MoveExternal`)	The path group is currently being moved, or was last moved, via a coupled movement.

68.1 Properties of MCGeo.State

The functionality provides the following properties:

- Value
- IsDisabled
- IsEnabled
- IsRunning
- IsSupended
- IsDisabledError
- IsEnabledError
- IsInactive
- OperatingMode
- TypeOfMotion
- Transitions (for a description, see MCGeo.State.Transitions [▶ 389])

68.1.1 Value

Description

Data type	Meaning
OperatingStates	Reads the operating state of the path group (for a detailed description, see section *Operating state* in MCGeo.State [▶ 378]).

Data type	Meaning
OperatingStates.Inactive	Inactive
OperatingStates.Disabled	Disabled
OperatingStates.Enabled	Enabled
OperatingStates.Running	Running
OperatingStates.Suspended	Suspended
OperatingStates.EnabledError	Error: enabled
OperatingStates.DisabledError	Error: disabled

Syntax

```
OpState := geoGroup.State.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.2 IsDisabled

Description

Data type	Meaning
Bool	Queries whether the group is in the **Disabled** operating state.

Syntax

```
When geoGroup.State.IsDisabled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.3 IsDisabledError

Description

Data type	Meaning
Bool	Queries whether the group is in the **Error: disabled** operating state.

Syntax

```
When geoGroup.State.IsDisabledError Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.4 IsEnabled

Description

Data type	Meaning
Bool	Queries whether the group is in the **Enabled** operating state.

Syntax

```
When geoGroup.State.IsEnabled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.5 IsEnabledError

Description

Data type	Meaning
Bool	Queries whether the group is in the **Error: enabled** operating state.

Syntax

```
When geoGroup.State.IsEnabledError Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.6 IsInactive

Description

Data type	Meaning
Bool	Queries whether the group is in the **Inactive** operating state.

Syntax

```
When geoGroup.State.IsInactive Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.7 IsRunning

Description

Data type	Meaning
Bool	Queries whether the group is in the **Running** operating state.

Syntax

```
When geoGroup.State.IsRunning Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.8 IsSuspended

Description

Data type	Meaning
Bool	Queries whether the group is in the **Suspended** operating state.

Syntax

```
When geoGroup.State.IsSuspended Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.9 OperatingMode.Value

Description

Data type	Meaning
Type MCOperatingModes	Reads the operating mode of the group (for a detailed description, see section Operating mode [▶ 378]).
MCOperatingModes.Inactive	The group is inactive, e.g. because it has not been activated.
MCOperatingModes.Normal	The group has been activated and is not in the error state.
MCOperatingModes.Error	The group is in the error state after an operating error.

Syntax

```
OpModeAxis := geoGroup.State.OperatingMode.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInInactiveMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *Inactive* operating mode.

Syntax

```
When geoGroup.State.OperatingMode.IsInInactiveMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInErrorMode

Description

Data type	Meaning
Bool	Queries whether the group is in the *Error* operating mode.

Syntax

```
When geoGroup.State.OperatingMode.IsInErrorMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInNormalMode

Description

Data type	Meaning
Bool	Queries whether the group is in the *Normal* operating mode.

Syntax

```
When geoGroup.State.OperatingMode.IsInNormalMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.1.10 TypeOfMotion.Value

Description

Operating state	Meaning
Type MCGeoTypesOfMotion	Motion type the path group is currently in (for a detailed description, see section MCGeo.State [▶ 380]).
MCGeoTypesOfMotion.MoveLinear	The path group is currently executing, or last executed, a linear movement.
MCGeoTypesOfMotion.MoveCircular2D	The path group is currently executing, or last executed, a 2D circular movement.
MCGeoTypesOfMotion.MoveCircular3D	The path group is currently executing, or last executed, a 3D circular movement.
MCGeoTypesOfMotion.MoveSpline	The path group is currently executing, or last executed, a spline movement.
MCGeoTypesOfMotion.MovePolynominal	The path group is executing a blending movement.
MCGeoTypesOfMotion.MovePtp	The path group is currently executing, or last executed, a point-to-point movement.
MCGeoTypesOfMotion.MoveOrientation	The path group is currently executing, or last executed, an orientation movement.
MCGeoTypesOfMotion.MoveExternal	The path group is currently being moved, or was last moved, via a coupled movement.

Syntax

```
TypeOfMotion := geoGroup.State. TypeOfMotion.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveLinear

Description

Data type	Meaning
Bool	Queries whether the group is in the motion type `Linear` .

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveLinear Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveCircular2D

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed a `2D circular movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveCircular2D Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveCircular3D

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed a `3D circular movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveCircular3D Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveSpline

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed a `spline movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveSpline Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMovePolynominal

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed a `blending movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMovePolynominal Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMovePtp

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed a `point-to-point movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMovePtp Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveOrientation

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed an `orientation movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveOrientation Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveExternal

Description

Data type	Meaning
Bool	Queries whether the group is currently executing or last executed a `coupled movement`.

Syntax

```
When geoGroup.State.TypeOfMotion.IsMoveExternal Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

68.2 Example task MCGeo.State

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample1 Autorun

    //Enable the path group
    If geoGroup.State.IsDisabled Then
        geoGroup.Power.Enable();
    End_if;
    When geoGroup.State.IsEnabled Continue;

    //A process with the group can be placed here, or other tasks move the
group
    //…

    // Disable the group
    If geoGroup.State.IsRunning Then //if the group is still moving
        geoGroup.MoveHalt.Start(); //Stop the group
    When geoGroup.State.IsEnabled Continue
    End if;

    geoGroup.Power.Disable() //Disable the group
    When geoGroup.State.IsDisabled Continue

End_Task;
```

69 MCGeo.State.Transitions

The **MCGeo.State.Transitions** functionality allows you to control state transitions of a path group for special applications.

With the exception of the Suspend() and Resume() functions, the state transitions of a group are usually triggered automatically by the motion commands and the behavior of the axis movement. Direct control of the state management is only required in exceptional cases, e.g. to respond in the event of an error. The **MCGeo.State.Transitions** functionality, as well as `Diagnostics.ClearErrors()`, allow operating state transitions to be triggered explicitly.

The **MCGeo.State** functionality allows you to query various information about a path group, such as operating mode, motion type, etc.

Basic and further information

In this context, also observe the following information:

– For a description of the other functionalities of MCGeo.State, see section MCGeo.State [▶ 378];

– For a detailed description of the structure of the MCX core, the JetSym online help (JetSym Motion and MotionControl > JetSym MotionControl > MotionControl basics > Operating concept > Overview: Operating concept)

– For basic information, Motion Control Application Note No. 54 (MCX motion control basics).

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) No. ??** *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

In addition, the JetSym online help provides useful information on path groups.

Operating mode

The operating mode (`OperatingMode`) indicates how the path group basically works. There are 3 operating modes:

Operating mode	Meaning
`Inactive` (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
`Normal` (normal)	The group can be used. Travel commands are executed in buffered mode (see MCGeo.Buffer [▶ 257]), i.e. all motion commands are added to a preliminary buffer and processed one after the other.
`Error` (error)	An operating error is pending. The operating state is retained, but no new functions and parameters can be passed.

Tab. 8: Operating modes of MCX path groups

Operating state

At any time, a group is in one, and only one, clearly defined operating state (`OperatingState`). If, for example, the group is in the **Running** operating state, the **Enabled** operating state is not active, although of course all composite axes have pulse enable.

In a particular state, always only a subset of all function calls is permitted. The state model for the axis is as follows:

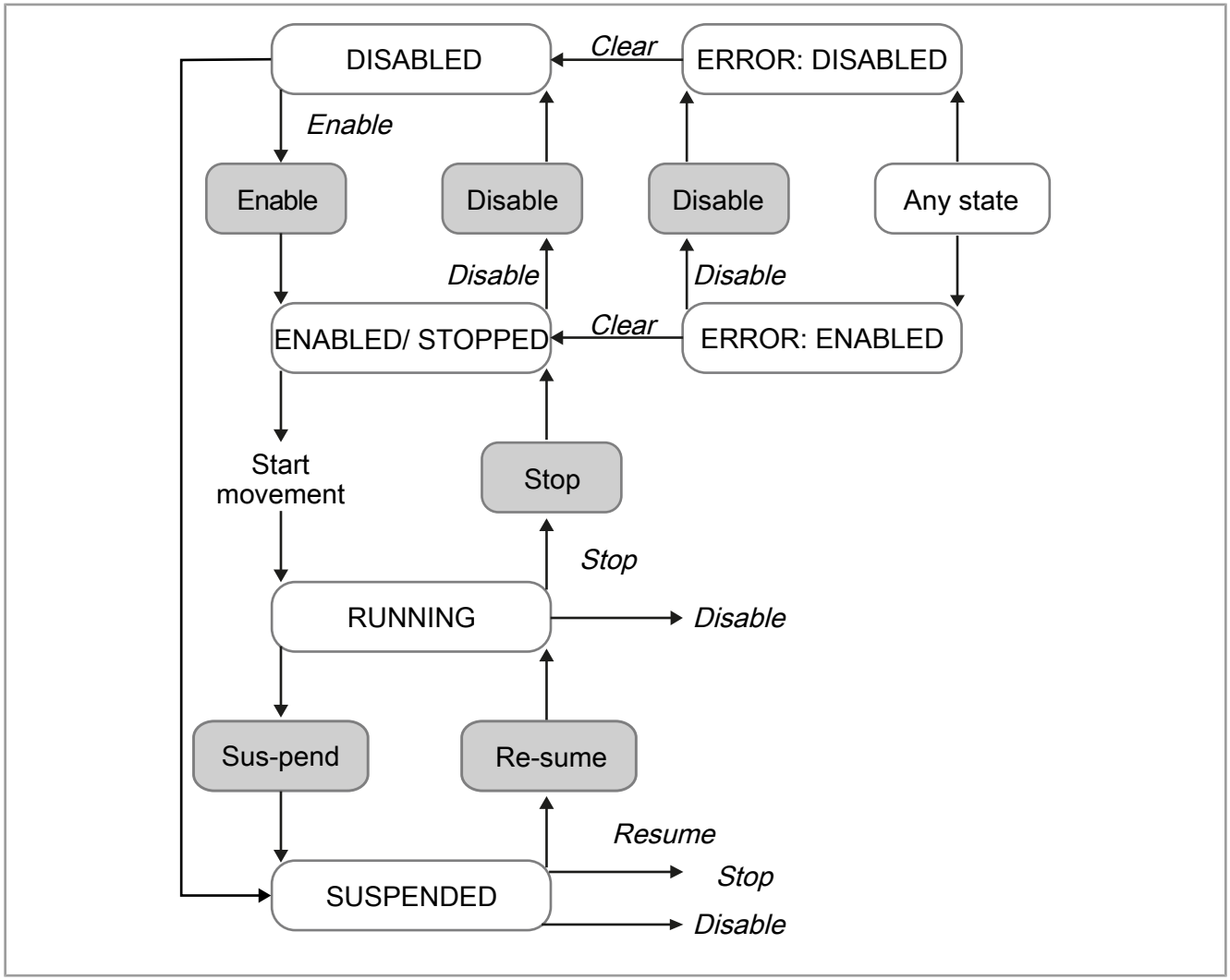

Fig. 14: State model for MCX groups

Operating state	Meaning
Inactive (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
Disabled (disabled)	– The controllers of the servo amplifiers of all composite axes are open (no pulse enable). – No torque is generated.
Enabled (enabled)	– The controllers of all composite axes are closed (pulse enable). – The setpoint interpolator is at standstill.
Running (running)	At least one axis is moving, i.e. the setpoint interpolator passes motion setpoints on to the servo amplifier.
Suspended (suspended)	– Corresponds to the **Enabled** operating state, with the exception that the target position is retained. A movement suspended with `MCGeo.State.Transitions.Suspend()` can be continued with the command `MCGeo.State.Transitions.Resume()` (state transition resume).
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control (error level 4).
DisabledError (Error: disabled)	An error has occurred. After the error response, the control of all axes is disabled (error levels 5 and 6).

Tab. 9: Operating states of path groups

Motion type

The motion type (`TypeOfMotion`) indicates which type of movement the setpoint generator is currently generating for the path group.

Querying the motion type is useful, for example, in a monitoring task in which the process is not known. Based on the motion type active before switching off, it is possible to control the restart correctly.

Operating state	Meaning
No movement	This motion type is active until the first movement of the group.
Straight line (`MoveLinear`)	The path group is currently executing, or last executed, a linear movement.
Circular movement 2D (`MoveCircular2D`)	The path group is currently executing, or last executed, a 2D circular movement.
Circular movement 3D (`MoveCircular3D`)	The path group is currently executing, or last executed, a 3D circular movement.
Spline (`MoveSpline`)	The path group is currently executing, or last executed, a spline movement.
Polynomial (`MovePolynomial`)	The path group is executing a blending movement.
Point-to-point (`MovePtp`)	The path group is currently executing, or last executed, a point-to-point movement.
Orientation (`MoveOrientation`)	The path group is currently executing, or last executed, an orientation movement.
Coupled movement (`MoveExternal`)	The path group is currently being moved, or was last moved, via a coupled movement.

69.1 Syntax for MCGeo.State.Transitions

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.State.Transitions.<function>(optional <parameter>);
```

69.2 Functions of MCGeo.State.Transitions

The functionality provides the following functions:

- Enable()
- Disable()
- Stop()
- ErrorDisable()
- ErrorStop()
- ErrorLockUserInput()
- Suspend()
- ErrorSuspend()
- Resume()

69.2.1 Enable()

The `State.Transistions.Enable()` function is identical to the `MCGeo.Power.Enable()` function. A detailed description of the function can be found in section MCGeo.Power.Enable() [▶ 372].

Syntax

```
geoGroup.State.Transitions.Enable();
```

69.2.2 Disable()

Depending on the parameter passed (see the following table), the function performs the same action as the commands `MCGeo.Power.Disable()` and `MCGeo.Power.Qickstop()`. For details, refer to the descriptions of MCGeo.Power [▶ 372].

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionDisableModes	Mode for executing the `Disable()` function
	MCStateTransitionDisable-Modes.ImmediatePulseBlocking	Performs the same function as the command `MCGeo.Power.Disable()` or `MCGeo MCGeo.Power.Disable(MCPowerDisable-Modes.Forced)`
	MCStateTransitionDisable-Modes.VelocityControlSlope	Performs the same function as the command `MCGeo.Power.QuickStop()`.

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), the function performs the same action as the command MCGeo.Stop() [▶ 400].

The difference is that passing a function parameter is mandatory with `MCGeo.State.Transition.Stop()`, whereas it is optional with `MCGeo.Stop()` (for details, see there).

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionStop-Modes	Two deceleration values are available.
	StopModes.ProgDeceleration	The deceleration last specified for the group is used for stopping.
	StopModes.MaxDeceleration	The maximum acceleration of the group is used as deceleration for stopping. It was either specified during commissioning in the motion setup under "Maximum deceleration" or programmed with the command `geoGroup.Mechanism.Limitations.MaxAcceleration`.

Syntax

```
geoGroup.State.Transitions.Stop(MCStateTransitionStopModes.ProgDecelera-
tion);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√		Enabled
Running	√	Running	Enabled
Suspended	√		Enabled
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

State monitoring

Various information is available for monitoring the positioning.

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis is stopped.

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //The setpoint generator of at least
one composite axis is generating a movement.
```

– The **ramp status** indicates which section of the movement 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;
```

– The **target window status** is not affected by this function.

69.2.4 ErrorDisable()

This function triggers an error in the path group that leads to the **Error: disabled** state. The error response can be set (see property `ErrorDisableMode`).

Parameters

The function has no parameters.

Syntax

```
geoGroup.State.Transitions.ErrorDisable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: disabled

Initial state	Allowed	Final state
Enabled	√	Error: disabled
Running	√	Error: disabled
Suspended	√	Error: disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

69.2.5 ErrorStop()

This function triggers an error in the group that leads to the **Error: enabled** state. The error response can be set (see property ErrorStopMode [▶ 398]).

Parameters

The function has no parameters.

Syntax

```
geoGroup.State.Transitions.ErrorStop();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: enabled
Enabled	√	Error: enabled
Running	√	Error: enabled
Suspended	√	Error: enabled
Error: enabled	-	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

69.2.6 ErrorLockUserInput()

This function triggers an operating error in the group.

Parameters

The function has no parameters.

Syntax

```
geoGroup.State.Transitions.ErrorLockUserInput();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

69.2.7 Suspend()

This function suspends a running group movement. The group is stopped either with the last programmed deceleration ramp or with the configured maximum deceleration. The state after stopping is not **Enabled** but **Suspended**. With the `Resume()` function, the suspended path movement of a group can be resumed and moved to the original target position. If the preliminary buffer is not yet empty, the further movements are appended.

The ramp setting (acceleration and deceleration) cannot be changed while the movements in the preliminary buffer are being traveled, as the precalculated movements are stored in the preliminary buffer. For this reason, the programmed deceleration for the `Suspend()` command uses the value that is also used for the path movements in the preliminary buffer. The value can only be changed after the movements in the preliminary buffer have been completed.

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionStopModes	Mode for the deceleration ramp of the `Suspend()` function
	MCStateTransitionStopModes.ProgDeceleration	Uses the last programmed deceleration of the group.
	MCStateTransitionStopModes.MaxDeceleration	Uses the maximum deceleration configured for the group.

Syntax

```
geoGroup.State.Transitions.Suspend(MCStateTransitionStopModes.ProgDecelera-
tion);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	X	Enabled

Initial state	Allowed	Final state
Running	√	Suspended
Suspended	–	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

69.2.8 ErrorSuspend()

This function triggers an error in the path group that leads to the **Error: enabled** state. The path movement is stopped with the programmed deceleration. The preliminary buffer is not cleared, so that the operating state after the error has been cleared is **Suspended** and the movement can be continued with Resume() [▶ 396].

Parameters

The function has no parameters.

Syntax

```
geoGroup.State.Transitions.ErrorSuspend();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: enabled
Enabled	√	Error: enabled
Running	√	Error: enabled
Suspended	√	Error: enabled
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

69.2.9 Resume()

The function continues a positioning that was suspended with the Suspend() [▶ 395] function. The originally specified target position is approached and further movements in the preliminary buffer are appended. The speed and acceleration values are also adopted from the original path movement.

The `Resume()` function only takes effect if the group is in the **Suspended** state.

In buffered mode, the preliminary buffer is not cleared.

Parameters

The function has no parameters.

Syntax

```
geoGroup.State.Transitions.Resume();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	X	Enabled
Running	–	Running
Suspended	√	Running
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

69.3 Properties of MCGeo.State.Transitions

The functionality provides the following properties:

– ErrorDisableMode
– ErrorStopMode
– ErrorLevel

69.3.1 ErrorDisableMode

Description

Data type	Meaning
MCStateTransitionDisableModes	Error response used when the `ErrorDisable()` function is triggered.
MCStateTransitionDisableModes.Velocity-ControlSlope	Stops the group with the quick stop deceleration. When 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;
ErrorResponse := geoGroup.State.Transitions.ErrorDisableMode;
```

State management

The property can be read and written in any state.

On read access to the property, 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 triggered.
MCStateTransitionErrorStop-Modes.ProgDeceleration	Stops the group with the programmed deceleration.
MCStateTransitionErrorStop-Modes.MaxDeceleration	Stops the group with the maximum deceleration.
MCStateTransitionErrorStop-Modes.AtBlockEnd	Stops the group at the end of the current movement. Processing of the preliminary buffer is stopped.

Syntax

```
geoGroup.State.Transitions.ErrorStopMode := MCStateTransitionStop-
Modes.MaxDeceleration;
ErrorResponse := geoGroup.State.Transitions.ErrorStopMode;
```

State management

The property can be read and written in any state.

69.3.3 ErrorLevel

Description

This property allows you to set the error level that the path group assumes in the event of a drive error. By default, this is error level 5 (stop with emergency stop ramp and **Error: disabled** operating state).

Data type	Meaning
MCStateTransitionErrorLevels	Error level of the path group in the event of a drive error. Detailed information on the differences between the error response levels can be found in the JetSym online help under *Error management*.
MCStateTransitionErrorLevels.Message	The group responds with error level 1: a message/warning is created.
MCStateTransitionErrorLevels.Channel-Stop	The group responds with error level 2: the command channel is locked.
MCStateTransitionErrorLevels.Cannel-StopClearBuffer	The group responds with error level 3: the command channel is locked and the preliminary buffer is cleared.
MCStateTransitionErrorLevels.Stop	The group responds with error level 4: an operational stop is executed.
MCStateTransitionErrorLevels.Quickstop (default)	The group responds with error level 5: an emergency stop is executed.
MCStateTransitionErrorLevels.PulsBlock-ing	The group responds with error level 6: the controller is disabled (pulse inhibit).

Syntax

```
geoGroup.State.Transitions.ErrorLevel := MCStateTransitionErrorLevels.Puls-
Blocking;
ResponseOnDriveError := geoGroup.State.Transitions.ErrorLevel;
```

State management

The property can be read and written in any state.

70 MCGeo.Stop

The **MCGeo.Stop** functionality allows you to stop a moving path group with a defined ramp.

Unlike the `MoveHalt.Start()` function, `Stop` clears the preliminary buffer, i.e. all travel blocks still in the buffer are deleted.

The functionality has no further functions or properties.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 50 – 52, 54 – 57** and the documents *Path group*, *Path group – synchronous axis* and *Path group path counter*. These are available for download on the Bucher Automation website.

70.1 Syntax for MCGeo.Stop

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Stop(optional <parameter 1>)
```

70.2 Functions of MCGeo.Stop

`MCTGeo.Stop` is a function of the group object.

70.2.1 Stop()

Parameters

Observe the following behavior:

– The function executes the stop command immediately. The parameter passed (see table) specifies the maximum deceleration for this.

– If no parameter is passed, the `StopMode` passed last is used.

– The group is stopped along the path.

Parameter	Data type	Meaning
StopModes	Enum StopModes (optional)	Two deceleration values are available.
	StopModes.ProgDeceleration	The deceleration last specified for the group is used for stopping (default).
	StopModes.MaxDeceleration	The maximum acceleration of the axis is used as deceleration for stopping. It was either specified during commissioning in the motion setup under Maximum deceleration or programmed with the command `geoGroup.Mechanism.Limitations.MaxAcceleration`.

Syntax

```
geoGroup.Stop(StopModes.ProgDeceleration);
geoGroup.Stop(StopModes.MaxDeceleration);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√		Enabled
Running	√	Running	Enabled
Suspended	√		Enabled
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning.

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis is stopped.

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //The setpoint generator of at least
one composite axis is generating a movement.
```

– The **ramp status** indicates which section of the movement 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;
```

– The **target window status** is not affected by this function.

71 MCManager (MCMgr)

The **MCMgr** functionality allows you to read information about the axis and group objects defined in the hardware tree of the project. This information can be used, for example, in the STX program for loop commands for all or some of the axis objects.

71.1 Syntax for MCMgr

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
MCMgr.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

71.2 Functions of MCMgr

The functionality provides the following functions:

- CheckVersions()
- GetDataSource()
- WriteLogToFile()
- GetAxisByID()
- GetAxisByName()
- GetTechnoByID()
- GetTechnoByName()
- GetGeoByID()
- GetGeoByName()

71.2.1 CheckVersions()

This function compares the version of the MCX kernel used with the minimum MCX kernel version required for the Motion API version used. In this way, you can check whether the operating system software of the control unit (with the MCX kernel it contains) is suitable for the functionalities of the Motion API version.

If the MCX kernel version is lower than the minimum required version, `false` is returned. In this case, the minimum OS version required for the control unit is output in the activated trace window of JetSym.

Parameters

This function has no parameters.

Syntax

```
if Not MCMgr.CheckVersions() then
    Taskbreak tInit;
end_if;
```

State management

The function can be read in any state.

71.2.2 GetDataSource()

Parameters

Parameter	Data type	Meaning
ShadowAxis	Pointer to MCAxis	Reads the axis object that is the data axis for the shadow axis passed to the function.

Syntax

```
DataAxis := MCMgr.GetDataSource(axShadow);
```

State management

The function can be read in any state.

71.2.3 WriteLogToFile()

This function writes the content of the MCX logger buffer to a text file that is stored under the name *MCLogger.dat* in the file system of the JetControl. This can be used to save the MCX actions during a specific program sequence (e.g. in the event of an error) in order to analyze them later. The file contains the logger messages encoded as numbers. To be able to analyze the data, you must transfer the file from the control unit to the PC and decode it into text.

Transferring the data file to the PC and decoding it

1. Transfer the file *MCLogger.dat* from the control unit to the computer using a tool with FTP communication or JetSym.
2. In JetSym, open the tab Hardware > MC-Global.
3. Select Motion setup > Status.
4. Make sure that the setup is deactivated (offline mode).
5. Click Open log file and select the storage location of the file *MCLogger.dat*.

   ⇨ A further dialog opens that allows you to save the decoded text file on the PC.
6. Follow the instructions.

⇨ The logger messages are displayed as text in the status.

Parameters

Parameter	Data type	Meaning
MaxLength	Integer (optional)	Defines the length of the data last recorded in the buffer [in bytes] that is to be transferred to the file. If no parameter or `0` is passed, the entire buffer memory recorded so far is transferred to the file. The length depends on the control unit type and can be up to 40 kbytes. The transfer is executed in the background and can take some time.

Syntax

```
MCMgr.WriteLogToFile(255);
```

State management

The function can be read in any state.

71.2.4 GetAxisByID()

The ObjectID of the axis is assigned by JetSym and must be unique in a project. In JetSym, it is displayed as Axis ID in the tooltip of the Hardware tree and on the Axis parameters page of the motion setup.

Parameters

Parameter	Data type	Meaning
ObjectID	Integer	Reads the axis object (return value of type `Pointer to MCAxis`) that is associated with the ObjectID passed.

Syntax

```
MCAxis := MCMgr.GetAxisByID(8);
```

State management

The function can be read in any state.

71.2.5 GetAxisByName()

The axis name must be unique in a project. It is assigned by JetSym when the axis is added to the Hardware tree, but can be changed.

Parameters

Parameter	Data type	Meaning
Name	String	Reads the axis object (return value of type `Pointer to MCAxis`) that is associated with the name string passed.

Syntax

```
MCAxis := MCMgr.GetAxisByName(`AX1´);
```

State management

The function can be read in any state.

71.2.6 GetTechnoByID()

The ObjectID of the technology group is assigned by JetSym and must be unique in a project. In JetSym, it is displayed as TechnoID in the tooltip of the Hardware tree and on the Properties page of the motion setup.

Parameters

Parameter	Data type	Meaning
ObjectID	Integer	Reads the group object (return value of type `Pointer to MCTechno`) that is associated with the ObjectID passed.

Syntax

```
MCTechGroup := MCMgr.GetTechnoByID(5);
```

State management

The function can be read in any state.

71.2.7 GetTechnoByName()

The group name must be unique in a project. It is assigned by JetSym when the group is added to the Hardware tree, but can be changed.

Parameters

Parameter	Data type	Meaning
Name	String	Reads the group object (return value of type `Pointer to MCTechno`) that is associated with the name string passed.

Syntax

```
MCTechno := MCMgr.GetTechnoByName(`AV1´);
```

State management

The function can be read in any state.

71.2.8 GetGeoByID()

The ObjectID of the path group is assigned by JetSym and must be unique in a project. In JetSym, it is displayed as GeoID in the tooltip of the Hardware tree and on the Properties page of the motion setup.

Parameters

Parameter	Data type	Meaning
ObjectID	Integer	Reads the group object of a path group (return value of type `Pointer to MCGeo`) that is associated with the ObjectID passed.

Syntax

```
MCGeoGroup := MCMgr.GetGeoByID(2);
```

State management

The function can be read in any state.

71.2.9 GetTechnoByName()

The group name must be unique in a project. It is assigned by JetSym when the group is added to the Hardware tree, but can be changed.

Parameters

Parameter	Data type	Meaning
Name	String	Reads the group object of a path group (return value of type `Pointer to MCTechno`) that is associated with the name string passed.

Syntax

```
MCGeo := MCMgr.GetGeoByName(`Path1´);
```

State management

The function can be read in any state.

71.3 Properties of MCMgr

The functionality provides the following properties:

- AxisCount
- AxisList
- TechnoCount
- TechnoList
- GeoCount
- GeoList
- FaultyObjectCount
- FaultyObject
- Name
- Version
- IsInitialized

71.3.1 AxisCount

Description

Data type	Meaning
Integer	Reads the number of axis objects of the project. This can be particularly helpful for loop commands in which all axes of a project are to execute the same function one after the other.

Syntax

```
//this sequence disables all MCX axes defined in the project
for nAxisIndex := 0 to mcMgr.AxisCount - 1 by 1 do
    mcMgr.AxisList[nAxisIndex].Power.Disable();
end_for;
```

State management

The property can be read in any state.

71.3.2 AxisList[]

Description

Data type	Meaning
Array of pointer to MCAxis	Accesses the MCX axes of the project with an index. This can be particularly helpful for loop commands in which all axes of a project are to execute the same function one after the other.

Syntax

```
//this sequence disables all MCX axes defined in the project
for nAxisIndex := 0 to mcMgr.AxisCount - 1 by 1 do
    mcMgr.AxisList[nAxisIndex].Power.Disable();
end_for;
```

State management

The property can be read in any state.

71.3.3 TechnoCount

Description

Data type	Meaning
Integer	Reads the number of technology group objects of the project. This can be particularly helpful for loop commands in which all technology groups of a project are to execute the same function one after the other.

Syntax

```
//this sequence activates all technology groups defined in the project
for nTechnoIndex := 0 to mcMgr.TechnoCount - 1 by 1 do
    mcMgr.TechnoList[nTechnoIndex].Activate();
end_for;
```

State management

The property can be read in any state.

71.3.4 TechnoList[]

Description

Data type	Meaning
Array of pointer to MCTechno	Accesses the technology groups of the project with an index. This can be particularly helpful for loop commands in which all technology groups of a project are to execute the same function one after the other.

Syntax

```
//this sequence activates all technology groups defined in the project
for nTechnoIndex := 0 to mcMgr.TechnoCount - 1 by 1 do
    mcMgr.TechnoList[nTechnoIndex].Activate();
end_for;
```

State management

The property can be read in any state.

71.3.5 GeoCount

Description

Data type	Meaning
Integer	Reads the number of path group objects of the project. This can be particularly helpful for loop commands in which all path groups of a project are to execute the same function one after the other.

Syntax

```
//this sequence activates all path groups defined in the project
for nGeoIndex := 0 to mcMgr.GeoCount - 1 by 1 do
    mcMgr.GeoList[nGeoIndex].Activate();
end_for;
```

State management

The property can be read in any state.

71.3.6 GeoList[]

Description

Data type	Meaning
Array of pointer to MCTechno	Accesses the path groups of the project with an index. This can be particularly helpful for loop commands in which all path groups of a project are to execute the same function one after the other.

Syntax

```
//this sequence activates all path groups defined in the project
for nGeoIndex := 0 to mcMgr.GeoCount - 1 by 1 do
    mcMgr.GeoList[nGeoIndex].Activate();
end_for;
```

State management

The property can be read in any state.

71.3.7 FaultyObjectCount

Description

Data type	Meaning
Integer	Reads the number of MCX objects of the project that are in the error state. Messages are not taken into account. For an application example, see property FaultyObjectList[] [▶ 408].

State management

The property can be read in any state.

71.3.8 FaultyObjectList[]

Description

Data type	Meaning
Array of pointer to MCTechno	Lists which MCX objects of the project report errors. Messages are not taken into account. This list allows you, for example, to read the error number of the MCX object determined.

Syntax

```
//this sequence checks whether errors are pending in the axes, the technology groups or path groups and determines the axis and group objects that indicate errors. The number of the last error is saved 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('Faulty 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 object ID concerned
                mcAx := mcMgr.GetAxisByID(mcObj.Info.ObjectID);
                // Read the error number
                AxErrorNo := mcAx.Diagnostics.GetErrorCode();
            end_if

            // Check whether the faulty object is a technology group
            if mcObj.Info.ClassID == MCClassIDs.Techno then
                // Get the techno object with the object ID concerned
                mcTec := mcMgr.GetTechnoByID(mcObj.Info.ObjectID);
                // Read the error number
                TecErrorNo := mcTec.Diagnostics.Software.GetErrorCode();
            end_if;

            // Check whether the faulty object is a path group
            if mcObj.Info.ClassID == MCClassIDs.Geo then
                // Get the geo object with the object ID concerned
                mcGe := mcMgr.GetGeoByID(mcObj.Info.ObjectID);
                // Read the error number
                GeoErrorNo := mcGe.Diagnostics.Software.GetErrorCode()
            end_if;

        end_for;
    end_if;

End_Task
```

State management

The property can be read in any state.

71.3.9 Name

Description

Data type	Meaning
String	Reads the type of MCX control unit used. The return value is a string (e.g. *JC-965EXT*).

State management

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 control unit. It is displayed in IP format.

State management

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 booting and is ready for operation. `True`: the MCX kernel was fully initialized during booting and is ready for operation `False`: Not all MCX axes were found on EtherCAT or their initialization failed. In the MCX logger, this is indicated by the following lines:	
	11:01:42:112	`Setup: Drives synchronous`
	11:01:42:112	`Setup: MC V. 1.2.114.2 successfully initialized and started`

Syntax

```
When mcMgr.IsInitialized continue;
```

State management

The property can be read in any state.

72 MCTechno.Activate, MCTechno.Deactivate

The **MCTechno.Activate** and **MCTechno.Deactivate** functionalities allow you to activate and deactivate a technology group. Changes to a technology group are permitted in all operating states of the group except **inactive**.

As soon as a technology group has been activated, all member axes are set to the **inactive** operating state. You can then only access these axes via the technology group in which they are integrated.

A technology group can only be activated and deactivated if all member axes of this group are either in the **enabled** or the **disabled** operating state and all member axes have the same operating state. Otherwise, an error message is output.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

72.1 Syntax for MCTechno.Activate, MCTechno.Deactivate

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Activate();
<group name>.Deactivate();
```

72.2 Functions of MCTechno.Activate, MCTechno.Deactivate

The functionality provides the following functions:

– Activate()
– Deactivate()

72.2.1 Activate()

This function activates a technology group, provided that all member axes are either in the **Disabled** or the **Enabled** state.

Parameters

The function has no parameters.

Syntax

```
techGroup.Activate();
```

State management

The following table applies to calling the function for the master and follower axis:

Initial state	Allowed	Final state
Inactive	√	Disabled/Enabled
Disabled	–	Disabled
Enabled	–	Enabled
Running	–	Running

Initial state	Allowed	Final state
Suspended	–	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Query in the program whether the new state has been reached before calling further commands for the axes.

```
When Not techGroup.State.IsInactive Continue;
```

Alternatively, you can also query the **IsEnabled** or **IsDisabled** operating state of the member axes – depending on which operating state the group had before the command was called.

72.2.2 Deactivate()

This function deactivates a technology group. After this command, the group is in the **inactive** state. The member axes of the group can be controlled independently again. They adopt the operating state of the technology group.

Parameters

The function has no parameters.

Syntax

```
techGroup.Deactivate();
```

State management

The following table applies to calling the function for the master and follower axis:

Initial state	Allowed	Final state
Inactive	–	Inactive
Disabled	√	Inactive
Enabled	√	Inactive
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Query in the program whether the new state has been reached before calling further commands for the axes.

```
When techGroup.State.IsInactive Continue;
```

Alternatively, you can also query the **IsEnabled** or **IsDisabled** operating state of the member axes – depending on which operating state the group had before the command was called.

73 MCTechno.Coupling

The **MCTechno.Coupling** functionality allows you to manage the coupling function of the technology group.

The functions assigned to the `Cam` property are described in detail in a separate section (MCTechno.Coupling.Cam [▶ 427]).

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

73.1 Syntax for MCTechno.Coupling

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Coupling.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

73.2 Functions of MCTechno.Coupling

The functionality provides 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 initiates the coupling of a follower axis in a technology group. The process requires the group to have been activated. Depending on the coupling mode, coupling is either executed directly, or a process is started that performs the coupling depending on various parameters of the master and follower axis. During this process, the coupling state is *IsCoupling*. At the end of the process, the coupling state is *IsCoupled*.

If the function is executed on the master axis, a message is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and is to be coupled.
CoupleMode	MCTechnoCoupleModes (optional)	Mode with which the coupling is to be executed. Familiarize yourself with the exact procedures used for the four coupling modes: – For electronic gearing: Application Note 050 – For a cam: Application Note 056
	MCTechnoCoupleModes Immediate	The coupling is executed directly and is active immediately.
	MCTechnoCoupleModes ImmediateSnap	The coupling is executed directly and is active immediately. – With **electronic gearing**, the behavior is identical to the *Immediate* coupling mode. – With a **cam**, the cam is shifted so that the setpoint matches the setpoint before coupling.
	MCTechnoCoupleModes Fast	The coupling takes place: – with acceleration via a coupling polynomial to the coupling position (see MasterPosition and FollowerPosition); – if necessary, with an increase of the coupling speed;
	MCTechnoCoupleModes Wait	The coupling takes place – with acceleration to the coupling position (see MasterPosition and FollowerPosition); – if necessary, by waiting for a modulo wrap-around in order to avoid an increase of the coupling speed;
MasterPosition	Double (optional)	For the *Fast* and *Wait* coupling modes: master axis position at which coupling is to be completed. The value must be greater than `0`.
MasterDirection	Directions (optional)	For the *Fast* and *Wait* coupling modes: direction of movement of the master axis. No coupling if the direction of movement is opposite.
	Directions.Positive	The master axis moves in the positive direction.
	Directions.Negativ	The master axis moves in the negative direction.
FollowerPosition	Double (optional)	For the *Fast* and *Wait* coupling modes: follower axis position at which coupling is to be completed.

Syntax

```
tecGroup.Coupling.Couple(Follower1,MCTechnoCoupleModes.Fast,1000.0,Direc-
tions.Positive,500.0);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Error: disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	√	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

73.2.2 Decouple()

This function initiates the decoupling of a follower axis in a technology group. The process requires the group to have been activated. Depending on the decoupling mode, decoupling is either executed directly, or a process is started that performs the decoupling depending on various parameters of the master and follower axis. During this process, the coupling state is *IsDecoupling*. At the end of the process, the coupling state is *IsDecoupled*.

If the function is executed on the master axis, a message is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and is to be decoupled.
DecoupleMode	MCTechnoDecouple-Modes (optional)	Mode with which the decoupling is to be executed. Familiarize yourself with the exact procedures used for the three decoupling modes: – For electronic gearing: Application Note 050; – For a cam: Application Note 056;
	MCTechnoDecouple-Modes Immediate_Stop	The decoupling is executed directly. The follower axis is stopped with the deceleration defined in the `Deceleration` parameter.
	MCTEechnoDecouple-Modes_KeepVelocity	The decoupling is executed directly. The follower axis continues to move at the speed it had at the time of decoupling (analogous to the `MoveVelocity()` command). If the master axis is in the stop ramp, the follower axis in the *Immediate_KeepVelocity* mode does not maintain its speed, but is stopped with the specified deceleration as in the *Immediate_Stop* mode.
	MCTechnoDecouple-Modes.AtMasterPosition_Stop	The follower axis is decoupled as soon as the master axis has reached the position value in the `MasterPosition` parameter. It is then stopped with the deceleration defined in the `Deceleration` parameter.
MasterPosition	Double (optional)	Decoupling position for the *AtMasterPosition_Stop* decoupling mode.

Parameter	Data type	Meaning
Deceleration	Double (optional)	Deceleration for stopping the follower axis after decoupling.

Syntax

```
tecGroup.Coupling.Decouple(Follower1, MCTechnoDecoupleModes.Immediate_Stop);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Error: disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	√	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

73.2.3 GetCoupleMasterPosition()

This function reads the master axis position last specified via the Couple function at which coupling is to be completed. (Only active in the *Fast* and *Wait* modes)

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the position of the master axis object of the technology group at which coupling is to be completed. (Only active in the *Fast* and *Wait* modes)

Syntax

```
MasterAxis_CouplePos := tecGroup.Coupling.GetCoupleMasterPosition(axMaster-
Axis);
```

State management

The function can be read in any state.

73.2.4 GetCoupleFollowerPosition()

This function reads the follower axis position last specified via the Couple function at which coupling is to be completed. (Only active in the *Fast* and *Wait* modes)

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the position of the follower axis object of the technology group at which coupling is to be completed. (Only active in the *Fast* and *Wait* modes)

Syntax

```
FollowerAxis_CouplePos := tecGroup.Coupling.GetCoupleFollowerPosition(Fol-
lower1);
```

State management

The function can be read in any state.

73.2.5 GetCoupleMasterDirection()

This function reads the master axis direction of movement last specified via the Couple function. (Only active in the *Fast* and *Wait* modes)

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the direction of movement of the master axis object of the technology group whose direction of travel is to be read. (Only active in the *Fast* and *Wait* modes)

Syntax

```
MasterAxis_Direction := tecGroup.Coupling.GetCoupleMasterDirection(axMaster-
Axis);
```

State management

The function can be read in any state.

73.2.6 GetDecoupleDeceleration()

This function reads the deceleration value that is used for decoupling in the *Immediate_Stop* and *AtMaster-Position_Stop* decoupling modes.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the deceleration value of the follower axis object of the technology group that is used for decoupling.

Syntax

```
DecouplingDecel := tecGroup.Coupling.GetDecoupleDeceleration(Follower1);
```

State management

The function can be read in any state.

73.2.7 GetDecoupleMasterPosition()

This function reads the decoupling position last passed in the Decouple function. (Only for the *AtMasterPosi-tion_Stop* decoupling mode.)

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the position of the follower axis object of the technology group whose master axis decoupling position is to be read. (Only for the *AtMasterPosition_Stop* decoupling mode)

Syntax

```
MasterAxis_DecouplePos := tecGroup.Coupling.GetDecoupleMasterPosition(Follower1);
```

State management

The function can be read in any state.

73.2.8 GetDecoupleMode()

This function reads the mode that is used for decoupling.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the decoupling mode that is used for the follower axis object of the technology group.

Syntax

```
AuskoppelModus := tecGroup.Coupling.GetDecoupleMode(Follower1);
```

State management

The function can be read in any state.

73.2.9 IsCoupling()

This function corresponds to the `IsCoupling` property of the follower axis (MCAxis.State.Techno.IsCoupling [▶ 215]).

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling state of a follower axis. `IsCoupling` is `true` if the axis has received the coupling command in the *fast* and *wait* coupling modes but has not yet reached the coupling point.

Syntax

```
When tecGroup.Coupling.IsCoupling(Follower1) Continue;
```

State management

The function can be read in any state.

73.2.10 IsCoupled()

This function corresponds to the `IsCoupled` property of the follower axis (MCAxis.State.Techno.IsCoupled [▶ 215]).

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling state of a follower axis. `IsCoupled` is `true` if the axis is coupled.

Syntax

```
When tecGroup.Coupling.IsCoupled(Follower1) Continue;
```

State management

The function can be read in any state.

73.2.11 IsDecoupling()

This function corresponds to the `IsDecoupling` property of the follower axis (MCAxis.State.Techno.IsDecoupling [▶ 214]).

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling state of a follower axis. `IsDecoupling` is `true` if the axis has received the decoupling command in the *AtMasterPosition_Stop* decoupling mode but has not yet reached the decoupling point.

Syntax

```
When tecGroup.Coupling.IsDecoupling(Follower1) Continue;
```

State management

The function can be read in any state.

73.2.12 IsDecoupled()

This function corresponds to the `IsDecoupled` property of the follower axis (MCAxis.State.Techno.IsDecoupled [▶ 214]).

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling state of a follower axis. `IsDecoupled` is `true` if the axis is decoupled.

Syntax

```
When tecGroup.Coupling.IsDecoupled(Follower1) Continue;
```

State management

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 position filtering in a technology group. An axis of the technology group and a filter coefficient can be passed as parameters.

Observe the following rules:

– If a **master axis** is selected, actual value filtering is performed. This is only useful if at least one follower axis is connected via actual value coupling.
– If a **follower axis** is selected, a filter in the setpoint path is activated.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Composite axis whose filter setting is to be changed or read.
Coefficient	Double	– Filter coefficient; the value is specified between `0` … `100`, where: `0`: no filter effect (default), `100`: maximum filter effect;

Syntax

```
tecGroup.Coupling.Filter.SetCoefficient(Technoaxis,22.4);
FilterCoeff := tecGroup.Coupling.Filter.GetCoefficient(Technoaxis);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

73.2.14 Gearing.Configure()

`Configure()` is a subfunction of the `Gearing` property. This function allows you to set the coupling parameters for a follower axis that is configured as electronic gearing.

The `Factor` and `Divisor` parameters allow you to set the gear ratio between the positions of the follower axis and the master axis.

Observe the following rules:

– A follower axis that is configured as **electronic gearing** is coupled directly when the technology group is activated.

  If this is not useful for the application, automatic coupling on activation of the group can be switched off with the `MCTechnoGearingActivationMode` parameter. This requires the group to have been activated. Automatic coupling then remains inactive until it is activated again.

– If the command is applied to a follower axis that is configured as a **cam**, no function is executed and no error message is output.

NOTICE



Risk of reparameterization of the master axis function

Applying the command to a master axis can lead to a reparameterization of the master axis function.

► Make sure to apply the command to follower axes only.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must belong to the technology group and whose coupling parameters are to be read or changed.
Factor	Double (optional)	Factor of the gear ratio.
Divisor	Double (optional)	Divisor of the gear ratio: Must be > 0.
ActivationMode	MCTechnoGearingActivationModes (optional)	Coupling status on activation of the technology group.
	MCTechnoGearingActivationModes.Coupled [Default]	Coupled on activation.
	MCTechnoGearingActivationModes.Decoupled	Decoupled on activation.

Syntax

```
tecGroup.Coupling.Gearing.Configure(Follower1, 1.0, 2.0, MCTechnoGearingAc-
tivationModes.Decoupled);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

73.2.15 Gearing.GetFactor()

`GetFactor()` is a subfunction of the `Gearing` property.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the factor of the gear ratio for a follower axis in a technology group that is configured as electronic gearing. If the command is applied to a master axis or to a follower axis configured as a cam, a meaningless value is returned and no error message is output.

Syntax

```
GearingFactor := tecGroup.Coupling.Gearing.GetFactor(Follower1);
```

State management

The function can be read in any state.

73.2.16 Gearing.GetDivisor()

`GetDivisor()` is a subfunction of the `Gearing` property. Applying the command to a master axis or to a follower axis configured as a cam returns a meaningless value and does not output an error message.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the divisor of the gear ratio for a follower axis in a technology group that is configured as electronic gearing.

Syntax

```
GetriebeDivisor := tecGroup.Coupling.Gearing.GetDivisor(Follower1);
```

State management

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 to a follower axis configured as a cam returns a meaningless value and does not output an error message.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Reads the coupling status on activation of the technology group for a follower axis that is configured as electronic gearing.

Syntax

```
GearingActivation := tecGroup.Coupling.Gearing.GetActiviationMode(Fol-
lower1);
```

State management

The function can be read in any state.

73.3 Properties of MCTechno.Coupling

The functionality provides the following properties:

– Cam (see MCTechno.Coupling.Cam [▶ 427])
– ExternalMaster.Offset
– ExternalMaster.Smoothing
– Filter (see the description of the subfunctions under Functions of MCTechno.Coupling [▶ 414])
– Gearing (see the description of the subfunctions under Functions of MCTechno.Coupling [▶ 414])

73.3.1 ExternalMaster.Offset

Description

This property is only useful if the master axis of a technology group is an external axis.

An external axis is characterized by the fact that it cannot be controlled by the setpoint generator of the MCX kernel, as it is merely the position of a position encoder. This property allows an offset to be added to this position before it is used as the master axis position in the technology group.

This command is ignored if the master axis of the technology group is not an external axis.

Data type	Meaning
Double	Reads or writes a position offset for the master axis position from an external axis.

Syntax

```
ExternalPosOffset := tecGroup.Coupling.ExternalMaster.Offset;
tecGroup.Coupling.ExternalMaster.Offset := 536,2;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

73.3.2 ExternalMaster.Smoothing

Description

The ExternalMaster.Smoothing property allows you to read the status or set the state of smoothing of the master axis position values for the external axis. If the property is activated, the position of the external axis is smoothed over the last four input values to compensate for possible fluctuations. This function can be used in addition to actual value smoothing.

This property is only useful if the master axis of a technology group is an external axis.

This command is ignored if the master axis of the technology group is not an external axis.

Data type	Meaning
MCTechnoSmoothingModes	Reads the status or sets the state of smoothing of the master axis position values for the external axis.
MCTechnoSmoothingModes.Disable	No smoothing is performed.
MCTechnoSmoothingModes.Standard	Smoothing is performed over the last 4 input values to compensate for possible fluctuations. This function can be used in addition to actual value smoothing.

Syntax

```
External_Smoothing := tecGroup.Coupling.ExternalMasterSmoothing;
tecGroup.Coupling.ExternalMaster.Smoothing :=
MCTechnoSmoothingModes.Standard;
```

State management

The property can be read in any state.

The following table applies to writing the property:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

74 MCTechno.Coupling.Cam

The **MCTechno.Coupling.Cam** functionality allows you to manage the creation and configuration of cams for a follower axis in a technology group.

A detailed description of the other functions and properties of the functionality can be found in the associated section MCTechno.Coupling [▶ 414].

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

APN 52 and APN 56 – Cams

Pay particular attention to the detailed information specifically on handling cams in **Application Notes 052 and 056**.

74.1 Syntax for MCTechno.Coupling.Cam

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Coupling.Cam.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

74.2 Functions of MCTechno.Coupling.Cam

The functionality provides the following functions:

- Create()
- DefineSegment()
- DefineSegmentAuto()
- Activate()
- ReadSegment()
- Evaluate()
- AnalyseSegment()
- AnalyseSegmentDerivates()

74.2.1 Create()

This function provides a cam. The cam is assigned an identification and memory space for the cam is provided in the control unit. These steps are required in order to define the segments of a cam, and to activate and couple the cam.

If the function is applied to the master axis or to a follower axis that is configured as electronic gearing, an error is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and for which a cam is to be created.
CamId	Integer	Defines a unique number for identifying this cam.
SegmentCount	Integer	Defines the number of segments of this cam.

Syntax

```
tecGroup.Coupling.Cam.Create(Z_Axis,1,3);
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.2 DefineSegment()

NOTICE



Risk of uncontrolled axis movements

Changes to the segment definitions take effect immediately.

- ► Avoid changing the segment definition of a cam that is currently active.
- ► If this is unavoidable, proceed with extreme caution.

This function defines, in relation to the master axis, the motion profile that is to apply to a section of the cam. The segment can only be defined if the cam has been created beforehand.

If the function is applied to the master axis or to a follower axis that is configured as electronic gearing, an error is output.

Difference between DefineSegment() and DefineSegmentAuto()

`DefineSegment()`: the polynomial sections are defined via the polynomial coefficients.

`DefineSegmentAuto()`: the polynomial sections are defined via initial and final conditions.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis 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 ID (`CamID`) to which the segment is to be assigned.
SegmentId	Integer	Unique number for identifying the segment in the cam selected with `CamId`.

Parameter	Data type	Meaning
SegmentType	MCTechnoCamSegment-Types (optional)	Type of curve shape of this segment in relation to the master axis position.
	MCTechnoCamSegment-Types.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegment-Types.Poly3rd	Curve shape: third-degree polynomial.
	MCTechnoCamSegment-Types.Poly5th	Curve shape: fifth-degree polynomial.
	MCTechnoCamSegment-Types.Poly7th	Curve shape: seventh-degree polynomial.
	MCTechnoCamSegment-Types.Sine	Curve shape: sine function.
SegmentStart	Double (optional)	Master axis position from which this segment is valid.
Option	MCTechnoCamOptions (optional)	Options for executing the segment.
	MCTechnoCamOp-tions.NoOption	The segment definition is adopted as specified.
	MCTechnoCamOp-tions.AutoAppend	This option is only available from the second segment onwards. It is intended to simplify the definition so that the follower axis offset of the segments does not have to be specified.
	MCTechnoCamOptions. UserDefinedMasterCam-Range	This option is only available in the first segment. It defines for the entire cam that the master axis range of the cam is not equal to the modulo range of the master axis. If the cam is to be stretched/compressed in the master axis direction during use, the `UserDefinedMasterRange` option must also be selected here, even if the specified `MasterCam-Range` corresponds to the modulo travel range of the master axis.
Masteroffset	Double (optional)	Offset of the master axis value that is added to the current master axis position.
FollowerOffset	Double (optional)	Offset of the follower axis value that is added to the function result.
Coefficient0	Double (optional)	Coefficients of the selected function according to the `Seg-mentType`. Not all coefficients are used for all `SegmentTypes`. **NOTICE! For more information, refer to the detailed information on the polynomial 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 option is used, the deviating master axis range must be defined here.

Syntax

```
tecGroup.Coupling.Cam.DefineSegment(xFollower,1,1,MCTechnoCamSegment-
Types.StraightLine, X1Offset, MCTechnoCamOptions.NoOption,-1.0 * X1Off-
set,Y1Offset,0.0,1.0)
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.3 DefineSegmentAuto()

NOTICE

Risk of uncontrolled axis movements

Changes to the segment definitions take effect immediately.

- ► Avoid changing the segment definition of a cam that is currently active.
- ► If this is unavoidable, proceed with extreme caution.

This function defines, in relation to the master axis, the motion profile that is to apply to a section of the cam. The segment can only be defined if the cam has been created beforehand.

If the function is applied to the master axis or to a follower axis that is configured as electronic gearing, an error is output.

Difference between DefineSegment() and DefineSegmentAuto()

`DefineSegment()`: the polynomial sections are defined via the polynomial coefficients.

`DefineSegmentAuto()`: the polynomial sections are defined via initial and final conditions.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis 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 ID (`CamID`) to which the segment is to be assigned.
SegmentId	Integer	Unique number for identifying the segment in the cam selected with `CamId`.
SegmentType	MCTechnoCamSegmentAutoTypes (optional)	Type of curve shape of this segment in relation to the master axis position.
	MCTechnoCamSegmentAutoTypes.AutoPoly1st	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegmentAutoTypes.AutoPoly3rd	Curve shape: third-degree polynomial.
	MCTechnoCamSegmentAutoTypes.AutoPoly5th	Curve shape: fifth-degree polynomial.
	MCTechnoCamSegmentAutoTypes.AutoPoly7th	Curve shape: seventh-degree polynomial.
SegmentStart	Double (optional)	Master axis position from which this segment is valid.

Parameter	Data type	Meaning
Option	MCTechnoCamOptions (optional)	Options for executing the segment.
	MCTechnoCamOptions.NoOption	The segment definition is adopted as specified.
	MCTechnoCamOptions.AutoAppend	This option is only available from the second segment onwards. It is intended to simplify the definition so that the follower axis offset of the segments does not have to be specified.
	MCTechnoCamOptions.UserDefinedMasterCamRange	This option is only available in the first segment. It defines for the entire cam that the master axis range of the cam is not equal to the modulo range of the master axis. If the cam is to be stretched/compressed in the master axis direction during use, the `UserDefinedMasterRange` option must also be selected here, even if the specified `MasterCamRange` corresponds to the modulo travel range of the master axis.
FollowerOffset	Double (optional)	Offset of the follower axis value that is added to the function result.
MasterRange	Double (optional)	Master axis distance covered by the segment.
FollowerRange	Double (optional)	Follower axis distance covered by the segment.
vStart	Double (optional)	Initial and final conditions for the first, second and third derivative of the follower axis position with respect to the master axis position. Not all parameters are used for all `SegmentTypes`. **NOTICE! For more information, refer to the detailed information on the derivatives in Application Note 052.**
vEnd	Double (optional)	
aStart	Double (optional)	
aEnd	Double (optional)	
jStart	Double (optional)	
jEnd	Double (optional)	
MasterCamRange	Double (optional)	If the `UserDefinedMasterCamRange` cam option is used, the deviating master axis range must be defined here.

Syntax

```
tecGroup.Coupling.Cam.DefineSegmentAuto(xFollower,1,1,MCTechnoCamSegmentAutoTypes.AutoPoli1st,X1Offset,MCTechnoCamOptions.NoOption,Y1Offset,0.0,1.0)
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.4 Activate()

This function activates a cam in an active technology group. Activation is only possible after the cam has been created. Activation in turn is a prerequisite for coupling the cam follower axis.

To change from one cam to another, activate the new (already defined) cam with a change condition. The follower axis remains coupled.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and for which a cam is to be activated.
CamId	Integer	Defines the cam ID (`CamID`) of the cam to be activated.
MasterOffset	Double (optional)	Defines the shift of the cam in the master axis direction.
FollowerOffset	Double (optional)	Defines the shift of the cam in the follower axis direction.
MasterFaktor	Double (optional)	Defines the scaling of the cam in the master axis direction.
FollowerFaktor	Double (optional)	Defines the scaling of the cam in the follower axis direction.
StartMode	MCTechnoCamStartModes (optional)	Defines the activation method.
	MCTechnoCamStart-Modes.Immediate	The cam becomes active immediately.
	MCTechnoCamStart-Modes.AtCamEnd	The cam becomes active as soon as the end of the running cam is reached.
	MCTechnoCamStart-Modes.AtMasterPosition	The cam becomes active when the master axis position defined in the `Position` parameter is reached.
	MCTechnoCamStart-Modes.AtFollowerPosition	The cam becomes active when the follower axis position defined in the `Position` parameter is reached.
Position	Double (optional)	Defines the master or follower axis position, depending on `StartMode`.

Syntax

```
tecGroup.Coupling.Cam.Activate(xFollower,1,0.0,0.0,1.0,1.0,MCTechnoCamStart-
Modes.Immediate,0.0);
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.5 ReadSegment()

This function reads the data of a cam segment of a defined cam. 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. In the case of 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 follower axis that is configured as electronic gearing, an error is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and for which the cam data is to be read.
CamId	Integer	Defines the ID (`CamID`) of the cam whose segment data is to be read.
SegmentId	Integer	Defines the ID of the segment whose data is to be read.

The structure of type `CamSegmentData` in which the data is stored has the following parameters:

Parameter	Data type	Meaning
SegmentType	MCTechnoCamSegmentTypes	Type of curve shape of this segment in relation to the master axis position.
	MCTechnoCamSegment-Types.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegmentTypes.Poly3rd	Curve shape: third-degree polynomial.
	MCTechnoCamSegmentTypes.Poly5th	Curve shape: fifth-degree polynomial.
	MCTechnoCamSegmentTypes.Poly7th	Curve shape: seventh-degree polynomial.
	MCTechnoCamSegmentTypes.Sine	Curve shape: sine function
	MCTechnoCamSegmentTypes.Au-toPoly1th	Curve shape: first-degree auto polynomial (linear)
	MCTechnoCamSegmentTypes.Au-toPoly3rd	Curve shape: third-degree auto polynomial.
	MCTechnoCamSegmentTypes.Au-toPoly5th	Curve shape: fifth-degree auto polynomial.
	MCTechnoCamSegmentTypes.Au-toPoly7th	Curve shape: seventh-degree auto polynomial.
SegmentStart	Double	Master axis position from which this segment is valid.
Option	Int	Options for executing the segment `0`: noOption `1`: AutoAppend `2`: UserDefinedMasterCamRange
Masteroffset	Double	Offset of the master axis value that is added to the current master axis position.
FollowerOffset	Double	Offset of the follower axis value that is added to the function result.
Coefficient	Array[0 .. 7] of Double	Coefficients of the function of this segment (always specified as polynomial coefficients).
MasterCam-Range	Double	For the first segment of a cam, the master axis range of the cam is returned here. For each further segment of a cam, the modulo range of the master axis is returned here.

Syntax

```
SegmentData := tecGroup.Coupling.Cam.ReadSegment(xFollower,1,2);
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.6 Evaluate()

This function reads the position and segment data of a cam. It allows you to determine the follower axis position and the associated segment that belong to a master axis position you pass.

The data is returned in a structure of type `CamEvaluateData`.

If the `AutoAppend` option is used in the `DefineSegment` or `DefineSegmentAuto` command, a correct value for the follower axis position can only be read after the cam has been started, as the follower axis offset is only calculated after the cam has been activated.

If the function is applied to the master axis or to a follower axis that is configured as electronic gearing, an error is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and for which the cam data is to be read.
CamId	Integer	ID (`CamID`) of the cam whose position and segment data is to be read.
MasterPosition	Double	Master axis position for which the corresponding data is to be determined.

The structure of type `CamEvaluateData` in which the data is stored has the following parameters:

Parameter	Data type	Meaning
FollowerPosition	Double	Setpoint position of the follower axis calculated by the cam at the master axis position passed.
SegmentNr	Integer	ID of the segment in which the position was determined.
MasterCamRange	Double	Master axis range of the cam.

Syntax

```
CamPosData := tecGroup.Coupling.Cam.Evaluate(xFollower,1,1721.2);
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.7 AnalyseSegment()

This function reads the data of a cam segment of a defined cam whose segments were created with the `AutoPoly` type. The function call returns an error if the cam was created with 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. speed, acceleration and jerk) in a segment. **NOTICE! Note that these positions are relative to the segment. To obtain the actual minimum or maximum position, the follower axis offset must be added.**

If the function is applied to the master axis or to a follower axis that is configured as electronic gearing, an error is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and for which the cam data is to be read.
CamId	Integer (optional)	ID (`CamID`) of the cam whose segment data is to be read.
SegmentId	Integer (optional)	ID of the segment whose data is to be read.

Parameter	Data type	Meaning
Option	MCTechnoCamAnalysisOptions (optional)	
	MCTechnoCamAnalysisOptions.AsCreated	Outputs the offsets in `CamAnalyseData.MasterOffest` and `CamAnalyseData.FollowerOffset` as they were defined by the user.
	MCTechnoCamAnalysisOptions.AsActivated	Outputs the offsets in `CamAnalyseData.MasterOffest` and `CamAnalyseData.FollowerOffset` including the additional activation offsets that the MCX calculates when activating the cam.

The structure of type `CamAnalyseData` in which the data is stored has the following parameters:

Parameter	Data type	Meaning
SegmentType	MCTechnoCamSegmentTypes	Type of curve shape of this segment in relation to the master axis position.
	MCTechnoCamSegmentTypes.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegmentTypes.Poly3rd	Curve shape: third-degree polynomial.
	MCTechnoCamSegmentTypes.Poly5th	Curve shape: fifth-degree polynomial.
	MCTechnoCamSegmentTypes.Poly7th	Curve shape: seventh-degree polynomial.
	MCTechnoCamSegmentTypes.Sine	Curve shape: sine function
	MCTechnoCamSegmentTypes.AutoPoly1st	Curve shape: first-degree auto polynomial (linear).
	MCTechnoCamSegmentTypes.AutoPoly3rd	Curve shape: third-degree auto polynomial.
	MCTechnoCamSegmentTypes.AutoPoly5th	Curve shape: fifth-degree auto polynomial.
	MCTechnoCamSegmentTypes.AutoPoly7th	Curve shape: seventh-degree auto polynomial.
	-1	An error has occurred, e.g. because the segment was not defined with the `AutoPoly` type. All other return values are then meaningless.
SegmentStart	Double	Master axis position from which this segment is valid.
Option	Integer	Options for executing the segment: `0`: noOption `1`: AutoAppend `2`: UserDefinedMasterCamRange
MasterOffset	Double	Offset of the master axis value that is added to the current master axis position.
FollowerOffset	Double	Offset of the follower axis value that is added to the function result.

Parameter	Data type	Meaning
MinValue	Double	`[0]`: minimum value of the 1st derivative of the position (speed). `[1]`: minimum value of the 2nd derivative of the position (acceleration). `[2]`: minimum value of the 3rd derivative of the position (jerk).
MaxValue	Double	`[0]`: maximum value of the 1st derivative of the position (speed). `[1]`: maximum value of the 2nd derivative of the position (acceleration). `[2]`: maximum value of the 3rd derivative of the position (jerk).

Syntax

```
SegmentData := tecGroup.Coupling.Cam.AnalyseSegment(xFollower,1,2);
```

State management

Provided that the technology group is active, the function can be called in any state.

74.2.8 AnalyseSegmentDerivates()

This function reads the data of a cam segment of a defined cam whose segments were created with the `AutoPoly` type. The function call returns an error if the cam was created with 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 derivatives of the position (i.e. speed, acceleration and jerk in relation to the master axis speed) in a segment.

If the function is applied to the master axis or to a follower axis that is configured as electronic gearing, an error is output.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Follower axis object that must be a member of the technology group and for which the cam data is to be read.
CamId	Integer (optional)	ID of the cam whose segment data is to be read.
SegmentID	Integer (optional)	ID of the segment whose data is to be read.

The structure of type `CamAnalyseDerivatesData` in which the data is stored has the following parameters:

Parameter	Data type	Meaning
SegmentType	MCTechnoCamSegmentTypes	Type of curve shape of this segment in relation to the master axis position.
	MCTechnoCamSegment-Types.StraightLine	Curve shape: linear (first-degree polynomial).
	MCTechnoCamSegment-Types.Poly3rd	Curve shape: third-degree polynomial.
	MCTechnoCamSegment-Types.Poly5th	Curve shape: fifth-degree polynomial.
	MCTechnoCamSegment-Types.Poly7th	Curve shape: seventh-degree polynomial.
	MCTechnoCamSegment-Types.Sine	Curve shape: sine function
	MCTechnoCamSegment-Types.AutoPoly1th	Curve shape: first-degree auto polynomial (linear).
	MCTechnoCamSegment-Types.AutoPoly3rd	Curve shape: third-degree auto polynomial.
	MCTechnoCamSegment-Types.AutoPoly5th	Curve shape: fifth-degree auto polynomial.
	MCTechnoCamSegment-Types.AutoPoly7th	Curve shape: seventh-degree auto polynomial.
	-1	An error has occurred, e.g. because the segment was not defined with the `AutoPoly` type. All other return values are then meaningless.
SegmentStart	Double	Master axis position from which this segment is valid.
Option	Integer	Options for executing the segment: `0`: noOption `1`: AutoAppend `2`: UserDefinedMasterCamRange
MasterOffset	Double	Offset of the master axis value that is added to the current master axis position.
FollowerOffset	Double	Offset of the follower axis value that is added to the function result.
MinValue	Array[0 .. 2] of Double	`[0]`: minimum value of the 1st derivative of the position (speed). `[1]`: minimum value of the 2nd derivative of the position (acceleration). `[2]`: minimum value of the 3rd derivative of the position (jerk).
MaxValue	Array[0 .. 2] of Double	`[0]`: maximum value of the 1st derivative of the position (speed). `[1]`: maximum value of the 2nd derivative of the position (acceleration). `[2]`: maximum value of the 3rd derivative of the position (jerk).

Parameter	Data type	Meaning
MasterCam-Range	Double	Master axis range of the cam.

Syntax

```
SegmentData := tecGroup.Coupling.Cam.AnalyseSegmentDerivates(xFollower,1,2);
```

State management

Provided that the technology group is active, the function can be called in any state.

75 MCTechno.Diagnostics

The **MCTechno.Diagnostics** functionality allows you to manage the error handling of a technology group.

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

The meaning of the error and message numbers is listed in the JetSym online help under "MotionControl errors".

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

75.1 Syntax for MCTechno.Diagnostics

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Diagnostics.<function or value access>(optional <parameter>);
```

75.2 Functions of MCTechno.Diagnostics

The functionality provides the following functions:

– ClearErrors()
– Software.ClearErrors()
– Software.GetErrorCode()
– Software.GetMessageCode()

75.2.1 ClearErrors(), Software.ClearErrors() – technology group

In the event of errors, these functions acknowledge an error state in the technology group. The entries in the error buffer are deleted.

Messages do not result in an error response and therefore do not need to be acknowledged. For messages, both functions clear the entries in the message buffer.

Parameters

These functions have no parameters.

Syntax

```
tecGroup.Diagnostics.ClearErrors();
tecGroup.Diagnostics.Software.ClearErrors();
```

ⓘ Info

Scope of the command

These commands clear all errors assigned to the group. These can be errors triggered by a group object as well as errors reported by the composite axes.

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state	
Disabled	–	Disabled	
Enabled	–	Enabled	
Running	–	Running	
Suspended	–	Suspended	
Error: enabled	√	Enabled	The system only changes to this state when the cause of the error has been eliminated. Otherwise, the error state may persist or be reached again immediately.
Error: disabled	√	Disabled	

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

75.2.2 Software.GetErrorCode()

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

Parameters

Parameter	Data type	Meaning
BufferIndex	Integer (optional)	`BufferIndex` passes the position in the error buffer whose error number is to be read. The maximum value of the `BufferIndex` parameter can be read via the `ErrorCount` property.

Syntax

```
tecGroup.Diagnostics.Software.GetErrorCode();
```

State management

The function can be read in any state.

75.2.3 Software.GetMessageCode()

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

Parameters

Parameter	Data type	Meaning
BufferIndex	Integer (optional)	`BufferIndex` passes the position in the message buffer whose message number is to be read. The maximum value of the `BufferIndex` parameter can be read via the `MessageCount` property.

Syntax

```
tecGroup.Diagnostics.Software.GetMessageCode();
```

State management

The function can be read in any state.

75.3 Properties of MCTechno.Diagnostics

The functionality provides 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 recorded since the `ClearErrors()` or `Software.Clear.Errors()` function was last executed or since the control unit was restarted.

Syntax

```
NumberOfErrors := tecGroup.Diagnostics.Software.ErrorCount;
```

State management

The property can be read in any state.

75.3.2 IsErrorPending, Software.IsErrorPending

Description

Data type	Meaning
Bool	Queries whether an error is pending in the MCX system. This is equivalent to comparing `ErrorCount` with 0.

Syntax

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

State management

The property can be queried in any state.

75.3.3 Software.MessageCount

Description

Data type	Meaning
Integer	Reads the number of messages recorded since the `ClearErrors()` or `Software.ClearErrors()` function was last executed or since the control unit was restarted.

Syntax

```
NumberOfMessages := tecGroup.Diagnostics.Software.MessageCount;
```

State management

The property can be read in any state.

75.3.4 IsMessagePending, Software.IsMessagePending

Description

Data type	Meaning
Bool	Queries whether a message is pending in the MCX system. This is equivalent to comparing `MessageCount` with `0`.

Syntax

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

State management

The property can be queried in any state.

75.4 Example task for MCTechno.Diagnostics

Here you will find an example task for a single axis that contains 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 the error variables used for display
    For i := 0 To 9 Do
        TecError[i] := 0;
    End_For;
    AcknowledgeError := False;

    When tecGroup.Diagnostics.IsErrorPending Continue;
```

```
        Taskbreak Automatic;
        // insert further error handling here

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

    // Errors have been eliminated
    When AcknowledgeError Continue;

        tecGroup.Diagnostics.ClearErrors();
        Taskrestart Automatic;
End_Loop;

End_Task;
```

76 MCTechno.Info

The **MCTechno.Info** functionality allows you to identify the group objects of the MotionControl.

This is information that the user has defined in the hardware manager and motion setup in the project.

76.1 Syntax for MCTechno.Info

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<technology group name>.Info.<property>;
```

76.2 Properties of the functionality

The functionality provides the following properties:

– Name
– ObjectID
– ClassID

76.2.1 Name

Description

Data type	Meaning
String	Queries 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;
```

Here, `TechnoName` contains the string `Techno`.

State management

The property can be read in any state.

76.2.2 ObjectID

Description

Data type	Meaning
Integer	Queries the control-internal number of the technology group. The property is normally only required in connection with applications that use the functions of the MC manager. A possible application is described in the corresponding section MCManager (MCMgr) [▶ 402].

Syntax

```
GroupID := TechnoGroup.Info.ObjectID;
```

State management

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. The property is normally only required in connection with applications that use the functions of the MC manager. A possible application is described in the corresponding section MCManager (MCMgr) [▶ 402].

Syntax

```
ClassID := tecGroup.Info.ClassID;
```

State management

The property can be read in any state.

76.3 Example task MCTechno.Info

Here you will find an example task for a single axis that contains various Motion API functions:

After the example task has been executed, the trace shows for all technology groups whether or not they are in the **running** state. The groups are listed with 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 by 1 do
        TraceRunningState(mcMgr.TechnoList[nTechnoIndex]);
    end_for;
end_task;
```

77 MCTechno.Mechanism.Slope

The **MCTechno.Mechanism.Slope** functionality 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 ramp
2. Travel at constant speed

   This phase only occurs if the speed `Speed` can be reached with the specified MovePtp parameters. Otherwise, the acceleration ramp changes directly into the deceleration ramp (triangular mode).
3. Deceleration ramp

In addition, there can also be no movement if the axis is at its target and no further positioning has been started yet.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

77.1 Syntax for MCTechno.Mechanism.Slope

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Mechanism.Slope.<value access>
```

77.2 Functions of MCTechno.Mechanism.Slope

The functionality provides the following functions:

– GetSlopeType()
– SetSlopeType()
– GetSpeed()
– SetSpeed()
– GetAcceleration()
– SetAcceleration()
– GetDeceleration()
– SetDeceleration()

77.2.1 GetSlopeType(), SetSlopeType()

These functions allow you to read or change the ramp shape in the acceleration and 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`. Only the pointer to an axis object that is a member of the technology group must be passed to it.

Parameters

Parameter	Data type	Meaning
Axis	pointer to MCAxis	Axis object (which must be a member of the technology group) whose ramp shape is to be read or changed
SlopeType	MCTech-noSlopeTypes	Shape of the acceleration and deceleration ramp.
	AxisSlopeTypes.Sine (default)	This setting calculates a movement that generates relatively little jerk, i.e. the movement is smoother. As the acceleration value passed to the motion command is only reached in the middle of the ramp, the ramp duration and length are greater than with the linear setting. In the acceleration phase, the movement is sinusoidal, and in the deceleration phase, it is sin²-shaped. The movement is jerk-limited.
	AxisSlopeTypes.Linear	This setting enables a movement with the lowest possible maximum acceleration at the same ramp time (and length) compared to the `sine` setting. In return, the jerk is infinite at the transition from standstill to the acceleration phase and from the acceleration phase to the phase with maximum speed. The movement is constant in the acceleration phase and sinusoidal in the deceleration phase. The movement takes place without jerk limitation.

Syntax

```
RampType := tecGroup.Mechanism.Slope.GetSlopeType(axMasterAxis);
tecGroup.Mechanism.Slope.SetSlopeType(axMasterAxis,MCTechnoSlopeTypes.Linear
);
```

State management

The function can be called in any state.

If the ramp shape is changed with `SetSlopeType()` during a positioning (**Running** state), this change only takes effect with the next positioning.

77.2.2 GetSpeed(), SetSpeed()

Description

Reads or writes the value for the maximum speed that is used for the position setpoint generation of 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 normally passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` or `MovePtp.NewSpeed()` and `MoveVelocity.NewSpeed()`, where a change is applied to a running movement.

This is different here: Regardless of the state in which the change of the value for `Speed` is passed, the new value only takes effect in the next movement.

Parameter	Data type	Meaning
Axis	pointer to MCAxis	Axis object that must be a member of the technology group and whose profile speed is to be read or changed.
Speed	Double	Maximum deceleration in the positioning profile.

Syntax

```
ProfileSpeed := tecGroup.Mechanism.Slope.GetSpeed(axMasterAxis);
tecGroup.Mechanism.Slope.SetSpeed(axMasterAxis, 3000.0);
```

State management

The function can be called in any state.

If the profile speed is changed with `SetSpeed()` during a positioning (**Running** state), this change only takes effect with the next positioning.

77.2.3 GetAcceleration(), SetAcceleration()

Description

Reads or writes the value for the maximum acceleration that is used for the position setpoint generation of 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 normally passed as a parameter in the `MovePtp.Start()` and `Move.Velocity.Start()` functions or in `MovePtp.NewAcceleration()` and `MoveVelocity.NewAcceleration()` .

Parameter	Data type	Meaning
Axis	pointer to MCAxis	Axis object that must be a member of the technology group and whose profile acceleration is to be read or changed.
Acceleration	Double	Maximum acceleration in the positioning profile.

Syntax

```
ProfileAcceleration := tecGroup.Mechanism.Slope.GetAcceleration(axMaster-
Axis);
tecGroup.Mechanism.Slope.SetAcceleration(axMasterAxis, 5500.0);
```

State management

The function can be called in any state.

If the profile acceleration is changed with `SetAcceleration()` during a positioning (**Running** state), this change only takes effect with the next positioning.

77.2.4 GetDeceleration(), SetDeceleration()

Description

Reads or writes the value for the maximum deceleration that is used for the position setpoint generation of 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 normally passed as a parameter in the functions `MovePtp.Start()` and `Move.Velocity.Start()` or `MovePtp.NewDeceleration()` and `MoveVelocity.NewDeceleration()`.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and whose profile deceleration is to be read or changed.
Deceleration	Double	Maximum deceleration in the positioning profile.

Syntax

```
ProfileDeceleration := tecGroup.Mechanism.Slope.GetDeceleration(axMaster-
Axis);
tecGroup.Mechanism.Slope.SetDeceleration(axMasterAxis,12000.0);
```

State management

The function can be called in any state.

If the profile deceleration is changed with `SetDeceleration()` during a positioning (**Running** state), this change only takes effect with the next positioning.

77.3 Properties of MCTechno.Mechanism.Slope

The functionality provides the following properties:

- SlopeState
- IsAccelerating
- IsAtConstantSpeed
- IsDecelerating
- IsStopped

77.3.1 SlopeState

Description

Data type	Meaning
MCTechnoSlopeStates	Reads which phase of the movement (acceleration, constant speed, deceleration or standstill) the path setpoint of the path group is currently in. The ramp status of the master axis is returned. If the technology group is inactive, **stopped** is returned. This property allows you, for example, to activate an action in the machine only once the group has left the acceleration phase.
MCTechnoSlopeStates.Accelerating	Group is in the acceleration ramp.
MCTechnoSlopeStates.AtConstand-Speed	Group is moving at constant speed.
MCTechnoSlopeStates.Decelerating	Group is in the deceleration ramp.
MCTechnoSlopeStates.Stopped	Group is at standstill.

Syntax

```
RampStatus := tecGroup.Mechanism.Slope.SlopeState;
```

State management

The property can be read in any state.

77.3.2 IsAccerlerating

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the master axis of the technology group is in the acceleration ramp.

Syntax

```
If tecGroup.Mechanism.Slope.IsAccelerating then …
```

State management

The property can be queried in any state.

77.3.3 IsAtConstantSpeed

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the master axis of the technology group is in the constant speed phase.

Syntax

```
If tecGroup.Mechanism.Slope.IsAtConstantSpeed then …
```

State management

The property can be queried in any state.

77.3.4 IsDecelerating

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the master axis of the technology group is in the deceleration ramp.

Syntax

```
If tecGroup.Mechanism.Slope.IsDecelerating then …
```

State management

The property can be queried in any state.

77.3.5 IsStopped

Description

Data type	Meaning
Bool	Queries whether the calculated setpoint of the master axis of the technology group is in the standstill phase.

Syntax

```
If tecGroup.Mechanism.Slope.IsStopped then …
```

State management

The property can be queried in any state.

78 MCTechno.Members

The **MCTechno.Members** functionality allows you to read the composition of a technology group.

For very special cases, you can also add or remove composite axes from within the STX program. As these cases are only recommended in exceptional situations, these commands are not described here. Please contact Bucher Automation AG if you want to use these functions.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

78.1 Syntax for MCTechno.Members

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Members.<value access>(optional <parameter 1>)
```

78.2 Properties of MCTechno.Members

The functionality provides the following properties:

- FollowerCount
- Follower[]
- FollowerCamCount
- FollowerCam[]
- FollwoerGearCount
- FollowerGear[]
- Master

78.2.1 FollowerCount

Description

Data type	Meaning
Integer	Reads the number of follower axis objects involved in the technology group.

Syntax

```
NumberOfFollowerAxesValue := tecGroup.Techno.Members.FollowerCount;
```

State management

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.2 Follower[n]

Description

Data type	Meaning
Pointer to MCAxis	Reads back the axis object that has the number *n* in the list of follower 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 management

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.3 FollowerCamCount

Description

Data type	Meaning
Integer	Reads the number of follower axis objects involved in the technology group that are defined as cams.

Syntax

```
CamAxisCount := tecGroup.Techno.Members.FollowerCamCount;
```

State management

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 that has the number *n* in the list of cam follower axes of the technology group. The list starts at `0` and goes up to `FollwerCamCount-1`.

Syntax

```
CamAxisObject := tecGroup.Members.FollowerCam[1].Info.ObjektID;
```

State management

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 follower axis objects involved in the technology group that are defined as electronic gearing.

Syntax

```
elGearAxisCount := tecGroup.Techno.Members.FollowerGearCount;
```

State management

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 that has the number *n* in the list of follower axes of the technology group defined as electronic gearing. The list starts at `0` and goes up to `FollwerGearCount-1`.

Syntax

```
elGearAxisObject := tecGroup.Members.FollowerGear[0].Info.ObjektID;
```

State management

The property can be queried in any state of the technology group, including the **inactive** state.

78.2.7 Master

Description

Data type	Meaning
Pointer to MCAxis	Reads back the axis object that is the master axis of the technology group.

Syntax

```
MasterAxisName := tecGroup.Members.Master.Info.Name;
```

State management

The property can be queried in any state of the technology group, including the **inactive** state.

79 MCTechno.MoveHalt

The **MCTechno.MoveHalt** functionality allows you to stop a moving composite axis of a technology group with a ramp.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

79.1 Syntax for MCTechno.MoveHalt

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveHalt.Start.(optional <parameter 1, parameter 2>);
```

79.2 Functions of MCTechno.MoveHalt

The functionality provides the following functions:

– Start()

79.2.1 Start()

The function immediately executes the stop command on the axis specified in the first parameter. The parameters passed define how the axis is to stop.

Observe the following rules:

– If the function is executed on the master axis, all axes stop according to their coupling type. All axes remain coupled.

– If the function is executed on a decoupled follower axis, this axis stops according to the parameters passed.

– For a coupled follower axis, the function acts on any superimposed movement. The movement of the follower axis resulting from the coupling with the master axis is not affected.

  To decouple and stop a coupled follower axis, use the `MCTechno.Coupling.Decouple(axFollowerAxis,MCTechnoDecoupleModes.Immediate_Stop,,DecelerationRamp)` function.

– The `Deceleration` parameter is entered in the user units predefined for the axis.

  Adjusting the maximum deceleration via this parameter changes the value in the axis object. If, for example, no value for the `Deceleration` parameter is passed with the next `MovePtp.Start` command, the value specified here is used.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameters

Parameter	Data type	Meaning
Axis	pointer to MCAxis	Axis object that must be a member of the technology group and is to be stopped.
MCAxisHaltMode	Enum MCAxisHaltModes (optional)	Behavior of the axis after it has been decelerated with a deceleration ramp:
	MCAxisHaltModes.Normal	After the stop ramp has been completed, the axis comes to a standstill in position control.
	MCAxisHaltModes.AtActualPosition	After the stop ramp has been completed, the axis moves back to the position where it was when the `MoveHalt.Start()` function was called. The return movement is executed with the parameters set for this axis object.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
tecGroup.MoveHalt.Start(axMasterAxis,MCTechnoHaltModes.Normal,50000);
tecGroup.MoveHalt.Start(axMasterAxis,MCTechnoHaltModes.AtActualPosition,50000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	–		Disabled
Enabled	–		Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

When querying whether stopping has been completed, only the **ramp status** should be taken into account.

The **operating state** of the group is a sum of the operating states of the composite axes. As soon as one member moves, the operating state changes to **Running**, and the **Enabled** state is only reached when all members are at standstill. The state can therefore be ambiguous depending on the application.

Ramp status of the master axis:

The group itself also has a ramp status. This is the ramp status of the master axis! For this reason, it must also be taken into account when querying the ramp status if the master axis is positioned. The ramp status of the master axis as a single axis always remains at **Stopped** while the group is active and therefore cannot be used.

```
// Master axis
tecGroup.MoveHalt.Start(axMasterAxis, MCTechnoHaltModes.Normal, 1000.0);
when tecGroup.Mechanism.Slope.IsStopped continue;
// Follower axis
tecGroup.MoveHalt.Start(axFollowerAxis, MCTechnoHaltModes.Normal, 1000.0);
when axFollowerAxis.Mechanism.Slope.IsStopped continue;
```

80 MCTechno.MovePtp

The **MCTechno.MovePtp** functionality allows you to start and parameterize point-to-point positioning of an axis of a technology group.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

80.1 Syntax for MCTechno.MovePtp

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MovePtp.<function or value access>(optional <parameter 1, parameter 2, parameter n>);
```

80.2 Functions of MCTechno.MovePtp

The functionality provides the following functions:

- Start()
- StartNewTargetPosition()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()
- GetPositioningMode()
- SetPositioningMode()
- GetTargetPosition()
- SetTargetPosition()
- GetTargetWindow()
- SetTargetWindow()
- StartSuperPose()
- StartNewTargetPositionSuperPose()

80.2.1 Start()

The function starts point-to-point positioning of an axis of a technology group. The axis starts moving immediately or applies the data to a running positioning.

Since all parameters required for positioning can be passed in a single command, this is the most compact way to fully parameterize and start a positioning movement.

If the function is executed on the master axis, all coupled follower axes move along according to their coupling function.

If the function is executed on a coupled follower axis, the axis concerned is first decoupled and then moved according to the parameters of the function. A follower axis that was originally coupled is not automatically coupled again after the movement has been completed.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Positioning in non-buffered mode (default)

Depending on whether the axis is moving or not, the behavior is as follows:

– Calling the function in the **enabled** state: The movement is started.

– Calling the function in the **running** state: The motion parameters are applied to the current OR next movement according to the following table.

Parameter	Data type	Meaning	Behavior when the function is called in the running state
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and is to be moved.	
Positioning-Mode	MCTechnoPositioningModes (optional)	Defines the positioning mode (see Get/SetPositioningMode [▶ 465]).	The mode takes effect immediately.
TargetPosition	Double (optional)	Defines the target position at which the axis is to stand after positioning has been carried out.	– If the new target position can be reached taking into account the maximum values for speed, acceleration, deceleration and jerk, it is approached directly using the new motion parameters. – Otherwise, the running positioning is completed and the newly programmed one follows directly.
Speed	Double (optional)	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration). The value must be greater than `0`.	If a new positioning is started 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 changed with the parameters of the acceleration ramp.

Parameter	Data type	Meaning	Behavior when the function is called in the running state
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	The speed ramps are executed according to the new values for acceleration and deceleration. – If the movement is currently in such a ramp, transitional forms occur that do not correspond to the set ramp type. – The transitions are always executed under monitoring of the jerk limit.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	
TargetWindow	Double (optional)	Defines the width of the target window for positioning.	The change of the target window is applied to the running positioning immediately.

Positioning phases

A positioning movement consists of the following phases:

1. Acceleration ramp
2. Constant speed

   This phase only occurs if the speed `Speed` can be reached with the specified MovePtp parameters. Otherwise, the acceleration ramp changes directly into the deceleration ramp (triangular mode).
3. Deceleration ramp

Ramp shapes

The ramp shapes of the acceleration and deceleration phase are specified by the axis parameter MCTechno.Mechanism.Slope.SetSlopeType (type MCTechnoSlopeTypes). There are two possible settings:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration,
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = sin²-shaped acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
tecGroup.MovePtp.Start(axMasterAxis,AxisPositioningModes.AbsNor-
mal,20000,500,1000,1000,0.5);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

– The **state of the group** indicates whether the setpoint generator of all composite axes is generating a movement or has been stopped.

```
When tecGroup.State.IsEnabled Continue; //the composite axes are in position
control and at standstill.
When tecGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

– If you want to query whether a defined position shortly before the target has been reached, the **target window** is used. The query is only possible for the relevant axis object of the composite axis.

```
When axMasterAxis.IsInTargetWindow Continue;
```

See also

- MCAxis.Buffer [▶ 31]
- MCAxis.Mechanism.Slope [▶ 135]

80.2.2 StartNewTargetPosition()

The function starts point-to-point positioning of a composite axis of the technology group. The axis starts moving immediately or adopts the data for changing a running positioning.

The function is a simplified variant of the `Start()` function, with which only the relevant position parameters are passed. Speed values and ramp settings for the axis are adopted.

This section describes the specifics of the `StartNewTargetPosition()` function. For all other information, please refer to the description of the Start() [▶ 458] function of the technology group.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning	Behavior when the function is called in the Running status
Axis	Pointer to MCAxis	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 [▶ 465]).	The mode takes effect immediately.
TargetWin-dow	Double (optional)	Defines the width of the target window for positioning.	The change of the target window is applied to the running positioning immediately.

Syntax

```
tecGroup.MovePtp.StartNewTargetPosition(axMasterAxis,AxisPositioning-
Modes.AbsNormal,20000,0.5);
```

80.2.3 NewSpeed()

The function passes a new speed value for point-to-point positioning of a composite axis of a technology group. The following applies:

– The value must be > `0`.

– If the axis is already moving (**Running** state), the new speed value is applied immediately.

  For a moving axis, the transition from the current to the new speed takes place with the acceleration set for the acceleration ramp. This is also the case if the speed decreases.

– The ramp type cannot be changed during positioning.

– If the ramp type *sin²* is set, the jerk limit is active for all transitions.

ⓘ Info

Bringing the axis to a standstill

To bring an axis to a standstill, only the `MoveHalt` or `Stop` commands may be used. It is **not** possible to stop an axis by passing a speed value of `0`.

Parameters

Parameter	Data type	Meaning	Behavior when the function is called in the Running state
Axis	Pointer to MCAxis	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 is to be positioned as long as it is not in the ramps (acceleration, deceleration).	The new speed value is applied immediately, taking into account the maximum values for acceleration, deceleration and jerk. The speed is changed with the parameters of the acceleration ramp.

Syntax

```
tecgroup.MovePtp.NewSpeed(axMasterAxis,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.4 NewAcceleration()

The function passes a new value for the maximum acceleration of point-to-point positioning of a composite axis of a technology group.

The following applies:

– An active acceleration ramp is not changed.

– The new value is applied with the next PtP positioning.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
tecGroup.MovePtp.NewAcceleration(axMasterAxis,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running

Initial state	Allowed	Final state
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.5 NewDeceleration()

The function passes a new value for the maximum deceleration of a point-to-point positioning of an MCX axis.

The following applies:

– The value is also used during a running PtP positioning if the target can still be reached with the new parameter.

– Otherwise, the value is applied with the next PtP positioning.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
tecGroup.MovePtp.NewDeceleration(axMasterAxis,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.6 GetPositioningMode(), SetPositioningMode()

Description

These functions allow you to read the currently valid positioning mode of a composite axis of a technology group or to set a new value for this axis.

The following applies:

– A new value takes effect with the next point-to-point positioning, even if the change is passed during a running positioning.
– After switching on, the last target position is calculated from the actual position at the time of switching on.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and is to be moved.

Parameter	Data type	Meaning
Positioning-Mode	MCTechnoPositioningMode	Type of PtP movement.
	MCTechnoPositioningMode.AbsNormal	Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the last and the new target position.
	MCTechnoPositioning-Mode.RelTarget	Moves the axis further by the distance passed in the `TargetPosition` parameter. The target position is calculated from the last target position plus the value in `TargetPosition`. The travel direction results from the sign of the `TargetPosition` parameter.
	MCTechnoPositioning-Mode.RelActual	Moves the axis further by the distance passed in the `TargetPosition` parameter. The target position is calculated from the actual position at the start of positioning plus the value in `TargetPosition`. The travel direction results from the sign of the `TargetPosition` parameter.
	MCTechnoPositioningMode.AbsModuloPos	Only valid for modulo axes. Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning is always **positive**. The modulo wrap-around can be passed if the new target position is smaller than the last target position.
	MCTechnoPositioningMode.AbsModuloNeg	Only valid for modulo axes. Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning is always **negative**. The modulo wrap-around can be passed if the new target position is larger than the last target position.
	MCTechnoPositioningMode.AbsModuloAuto	Only valid for modulo axes. – Positioning is started with the axis stopped: Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position **by the shortest path**. The travel direction of the positioning results from the shortest path to the target. – Positioning is started with the axis moving: Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the travel direction of the running positioning.
	MCTechnoPositioningMode.AbsModuloKeepDirection	Only valid for modulo axes. Moves the axis to the position passed in the `TargetPosition` parameter. This position is approached relative to the reference position. The travel direction of the positioning results from the travel direction of the last positioning.

Syntax

```
MasterAxisPositioningMode := tecGroup.MovePtp.GetPositioningMode(axMaster-
Axis);
tecGroup.MovePtp.SetPositioningMode(axFollowerAxis,MCTechnoPositioningModes.
RelTarget);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.7 GetTargetPosition(),SetTargetPosition()

Description

These functions allow you to read the currently valid value of `TargetPosition` of a composite axis of a technology group or to write a new value for this axis. If the function is called in the **Running** state, the motion parameters take effect when the next point-to-point positioning is started.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and is to be moved.
TargetPosition	Double	Property for determining the target position depending on the `PositioningMode` property.

Syntax

```
Target := tecGroup.MovePtp.GetTargetPosition(axFollowerAxis);
tecGroup.MovePtp.SetTargetPosition(axMasterAxis,12000);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running

Initial state	Allowed	Final state
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.8 GetTargetWindow(), SetTargetWindow()

Description

These functions allow you to read the currently valid value of the target window setting of a composite axis of a technology group or to write a new value for this axis. It takes effect when the next point-to-point positioning is started.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 area of the target position of a positioning.

Syntax

```
Targetwindow := tecGroup.MovePtp.GetTargetWindow(axMasterAxis);
tecGroup.MovePtp.SetTargetWindow(axFollowerAxis,0.7);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.9 StartSuperPose()

ⓘ Info

APN 51 – Superimposed positioning

Please refer to the detailed information in **Application Note 051** on the function of the superimposed movement of a follower axis in electronic gearing.

The function starts a superimposed point-to-point positioning of a follower axis of a technology group that is coupled via electronic gearing.

The following behavior applies:

– If the function is executed on a coupled follower axis that is configured as electronic gearing, the superimposed movement is started.

– If the function is executed on a decoupled follower axis that is configured as electronic gearing, or on the master axis, a movement is started for this axis that behaves as if it had been executed with the Start() [▶ 458] function. For more information, please refer to the linked section.

– If the function is executed on a follower axis that is configured as a cam, a message is output.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 [▶ 465]).
TargetPosition	Double (optional)	Defines the target position at which the axis is to stand after positioning has been carried out.
Speed	Double (optional)	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration). The value must be greater than `0`.
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.
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 management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

80.2.10 StartNewTargetPositionSuperPose()

ⓘ Info

APN 51 – Superimposed positioning

Please refer to the detailed information in **Application Note 051** on the function of the superimposed movement of a follower axis in electronic gearing.

The function is an abbreviated variant of the `StartSuperPose()` function. Both functions start a superimposed point-to-point positioning of a follower axis of a technology group that is coupled via electronic gearing. With this variant, only the parameters relevant to the position are passed. The values for speed and ramp settings are adopted as last set for the axis.

The following behavior applies:

– If the function is executed on a coupled follower axis that is configured as electronic gearing, the superimposed movement is started.

– If the function is executed on a decoupled follower axis that is configured as electronic gearing, or on the master axis, a movement is started for this axis that behaves as if it had been executed with the StartNewTargetPosition() [▶ 461] function. For more information, please refer to the linked section.

– If the function is executed on a follower axis that is configured as a cam, a message is output.

Parameters

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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() [▶ 465].

Parameter	Data type	Meaning
TargetWindow	Double (optional)	Defines the width of the target window for positioning.

Syntax

```
tecGroup.MovePtp.StartTargetPositionSuperPose(axElGetriebe,AxisPositioning-
Modes.AbsNormal,200,0.5);
```

Details on state management and state monitoring are identical to the Start() [▶ 458] function.

81 MCTechno.MoveVelocity

The **MCTechno.MoveVelocity** functionality allows you to start and parameterize continuous positioning of an MCX axis in a technology group.

ⓘ Info

Further information

For procedures and sample projects on working with axes, please refer to **Motion Control Application Notes (APN) Nos. 54, 55 and 57**. These are available for download on the Bucher Automation website.

81.1 Syntax for MCTechno.MoveVelocity

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.MoveVelocity.<function or value access>(optional <parameter 1,
parameter 2, parameter n>);
```

81.2 Functions of MCTechno.MoveVelocity

The functionality provides the following functions:

- Start()
- Reverse()
- NewSpeed()
- NewAcceleration()
- NewDeceleration()
- GetDirection()
- SetDirection()
- StartSuperPose()
- ReverseSuperPose()

81.2.1 Start()

The function starts continuous positioning of an axis in a technology group. The axis starts moving immediately or applies the data to a running positioning.

The following applies:

- If the function is executed on the **master axis**, all coupled follower axes move along according to their coupling function.
- If the function is executed on a coupled **follower axis**, the axis concerned is first decoupled and then moved according to the parameters of the function. A follower axis that was originally coupled is not automatically coupled again after the movement has been completed.
- The axis is accelerated with the current or programmed ramp and moves at the current or programmed value for `Speed` until it is stopped by a halt or stop command. Continuous positioning is position-controlled, so that a position following error is also monitored during the movement.

Parameters

Observe the following rules:

- All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Positioning in non-buffered mode

Depending on whether the axis is moving or not, the behavior is as follows:

– Calling the function in the **Enabled** state: The movement is started.

– Calling the function in the **Running** state: The motion parameters are applied to the current OR next movement according to the following table.

Parameter	Data type	Meaning	Behavior when the function is called in the Running state
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and is to be moved.	
Direction	Directions (optional) – Directions.Positive – Directions.Negative	Defines the direction of movement (see GetDirection(), SetDirection() [▶ 478]).	The new direction parameters are applied directly. If the new direction is opposite to the direction of the current movement, a reversing operation is executed with the acceleration ramps.
Speed	Double (optional)	Defines the speed at which the axis is to be positioned as long as it is not in the ramps (acceleration, deceleration). The value must be greater than `0`.	If a new positioning is started 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 changed with the parameters of the acceleration ramp.
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	The speed ramps are executed according to the new values for acceleration and deceleration. – If the movement is currently in such a ramp, transitional forms occur that do not correspond to the set ramp type. – The transitions are always executed under monitoring of the jerk limit.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning with the next `MoveHalt.Stop()` command if no other value is passed there. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.	

Ramp shapes

The ramp shapes of the acceleration and deceleration phase are specified by the axis parameter `MCTechno.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (see MCAxis.Mechanism.Slope [▶ 135]). There are two possible settings:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration,
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = sin²-shaped acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.Start(axMasterAxis,Directions.Positive,
Speed1,5000,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Running
Running	√	Running	Running
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

The **state of the group** indicates whether the setpoint generator of all composite axes is generating a movement or the group has been stopped.

```
When tecGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
When tecGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

81.2.2 Reverse()

The function reverses the direction of a running, superimposed continuous positioning of a follower axis in a technology group that is coupled via electronic gearing. This means that the direction of movement is reversed and the composite axis is moved in the opposite direction at the speed last set.

The following applies:

– The change of direction of movement is executed with the acceleration ramp last set.

– If the axis is in the **Enabled** state, the command has no effect.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.Reverse(axMasterAxis);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	–		Enabled
Running	√	Running	Running
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

81.2.3 NewSpeed()

The function passes a new speed value for continuous positioning of a composite axis. The following applies:

– The value must be > `0`.

– If the axis is already moving (**Running** state), the new speed value is applied immediately.

For a moving axis, the transition from the current to the new speed takes place with the values set for the acceleration ramps. This is also the case if the speed decreases.

– If the axis is at standstill (**Enabled** state), the value is only applied with the next positioning.
– The ramp type cannot be changed during positioning.
– If the ramp type *sin²* is set, the jerk limit is active for all transitions.

ⓘ Info

Bringing the axis to a standstill

To bring an axis to a standstill, only the `MoveHalt` or `Stop` commands may be used. It is **not** possible to stop an axis by passing a speed value of `0`.

Parameters

Parameter	Data type	Meaning	Behavior when the function is called in the Running state
Axis	Pointer to MCAxis	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 is to be positioned as long as it is not in the ramps (acceleration, deceleration).	If a new positioning is started 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 changed with the parameters of the acceleration ramp.

Syntax

```
tecGroup.MoveVelocity.NewSpeed(axMasterAxis,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.
– the state in which calling the function is not recommended but does not trigger an error.
X the state in which calling the function is not allowed and triggers an error.

81.2.4 NewAcceleration()

The function passes a new value for the maximum acceleration of continuous positioning of a composite axis.

The following applies:

– An active acceleration ramp is not changed.
– The new value is applied with the next continuous positioning.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
tecGroup.MoveVelocity.NewAcceleration(axMasterAxis,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

81.2.5 NewDeceleration()

The function passes a new value for the maximum deceleration of continuous positioning of a composite axis.

The following applies:

– The value is also used during a running positioning if the target can still be reached with the new parameter.

– Otherwise, the value is applied with the next positioning.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 reached in the stop ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Syntax

```
tecGroup.MoveVelocity.NewDeceleration(axMasterAxis,2000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

81.2.6 GetDirection(), SetDirection()

Description

These functions allow you to read the currently valid direction of a composite axis or to set a new value for this axis. The following applies:

– The value takes effect with the next continuous positioning.

– This is also the case if the change is passed during a running positioning.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and is to be moved.
Direction	Directions (optional)	Direction of the continuous movement.
	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(axMasterAxis);
tecGroup.MoveVelocity.SetDirection(axMasterAxis,Directions.Negative);
```

State management

The function can be read in any state.

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running

Initial state	Allowed	Final state
Suspended	X	Suspended
Error: enabled	√	Error: enabled
Error: disabled	√	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

81.2.7 StartSuperPose()

ⓘ Info

APN 51 – Superimposed positioning

Please refer to the detailed information in **Application Note 051** on the function of the superimposed movement of a follower axis in electronic gearing.

The function starts a superimposed continuous positioning of a follower axis in a technology group that is coupled via electronic gearing. The axis starts moving immediately or applies the data to a running positioning.

The following applies:

– If the function is executed on a **coupled follower axis** that is configured as electronic gearing, the superimposed movement is started.

– If the function is executed on a **decoupled follower axis** that is configured as electronic gearing, or on the **master axis**, the axis concerned starts a movement that behaves identically to execution with the Start() [▶ 472] function.

– If the function is executed on a **follower axis** that is configured as a cam, a message is output.

Parameters

Observe the following rules:

– All parameters are specified in the units that the user has predefined for the axis.

– The parameters refer to the axis object, i.e. they form a property of the axis. If no value is passed for a parameter, motion control uses the value of the parameter that was passed last.

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	Axis object that must be a member of the technology group and is to be moved.
Direction	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 is to be positioned as long as it is not in the ramps (acceleration, deceleration). The value must be greater than `0`.

Parameter	Data type	Meaning
Acceleration	Double (optional)	Defines the maximum acceleration to be reached in the start ramp during positioning. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.
Deceleration	Double (optional)	Defines the maximum deceleration to be reached in the stop ramp during positioning with the next `MoveHalt.Stop()` command if no other value is passed there. Possible ramp types: – *sin²*: The maximum deceleration is reached in the middle of the ramp. – *linear*: The ramp is run with the maximum deceleration throughout.

Ramp shapes

The ramp shapes of the acceleration and deceleration phase are specified by the axis parameter `MCTechno.Mechanism.Slope.SlopeType` (type `AxisSlopeTypes`) (see SlopeType [▶ 137]). There are two possible settings:

– `AxisSlopeTypes.Linear`

  Acceleration phase = constant acceleration,
  deceleration phase = sinusoidal acceleration

– `AxisSlopeTypes.Sine`

  Acceleration phase = sinusoidal acceleration,
  deceleration phase = sin²-shaped acceleration

ⓘ Info

Jerk limit

The sin and sin² ramps are checked for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.StartSuperPose(axElGetriebe,Directions.Nega-
tive,200,1000,1000);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√	Running	Enabled
Running	√	Running	Enabled
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the positioning:

The **state of the group** indicates whether the setpoint generator of all composite axes is generating a movement or the group has been stopped.

```
When tecGroup.State.IsEnabled Continue; //the group is in position control
and at standstill.
When tecGroup.State.IsRunning Continue; //the setpoint generator is
generating a movement.
```

81.2.8 ReverseSuperPose()

The function reverses the direction of a running, superimposed continuous positioning of a follower axis in a technology group that is coupled via electronic gearing. This means that the direction of movement is reversed and the composite axis is moved in the opposite direction at the speed last set.

The following applies:

– The change of direction of movement is executed with the acceleration ramp last set.

– If the axis is not in continuous positioning, the command has no effect.

– If the function is executed on a **decoupled follower axis** that is configured as electronic gearing, or on the **master axis**, the axis concerned starts a movement that behaves identically to execution with the Reverse() [▶ 185] function.

– If you attempt to execute the function on a **follower axis** that is configured as a cam, a message is output.

ⓘ Info

APN 51 – Superimposed positioning

Please refer to the detailed information in **Application Note 051** on the function of the superimposed movement of a follower axis in electronic gearing.

Parameters

Parameter	Data type	Meaning
Axis	Pointer to MCAxis	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 for compliance with the maximum permissible jerk (parameter `MCAxis.Mechanism.Limitations.MaxJerk`) and adjusted accordingly if necessary.

Syntax

```
tecGroup.MoveVelocity.ReverseSuperPose(axElGetriebe);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled

Initial state	Allowed	During execution	Final state
Enabled	–		Enabled
Running	√	Running	Running
Suspended	X		Suspended
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

82 MCTechno.Power

The **MCTechno.Power** functionality allows you to enable or disable the control and thus the torque generation of a technology group. All axes involved in the group are set to the new operating state simultaneously.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

82.1 Syntax for MCTechno.Power

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Power.<function>(optional <parameter>;
```

82.2 Functions of MCTechno.Power

The functionality provides the following functions:

– Enable()
– Disable()
– Quickstop()

82.2.1 Enable()

The function activates the controllers of the servo amplifiers of all composite axes and enables the voltage to the motor. A technology group can only be enabled if all composite axes are ready for operation.

Parameters

The function has no parameters.

Syntax

```
tecGroup.Power.Enable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Enabled
Enabled	–	Enabled
Running	–	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, query in the program whether the new state has been reached before calling further commands for the group.

```
When tecGroup.State.IsEnabled Continue;
```

82.2.2 Disable()

The function disables the servo amplifiers of all composite axes for torque generation (pulse inhibit). The motors are de-energized immediately and coast to a stop if no brake becomes active.

Parameters

Parameter	Data type	Meaning
DisableModes	MCPowerDisableMode (optional)	Mode for executing the `Disable()` function.
	MCPowerDisableMode.Forced (default)	The pulse inhibit always takes effect immediately. This mode is also active if no parameter is specified.
	MCPowerDisableMode.StateCheck	The pulse inhibit only takes effect if the group is in the **Enabled** operating state. In all other states, an operating error is output.

Syntax

```
tecGroup.Power.Disable();
tecGroup.Power.Disable(MCPowerDisableModes.Forced);
tecGroup.Power.Disable(MCPowerDisableModes.StateCheck);
```

State management

The following table applies to writing the function:

The following table applies to calling without a parameter or with the `MCPowerDisableModes.Forced` parameter:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	√	Disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

The following table applies to calling with the `MCPowerDisableModes.StateCheck` parameter:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, before calling further commands for the group, query in the program whether the new state has been reached, as follows:

```
When 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()

The function stops all composite axes with the quick stop deceleration and disables the servo amplifiers once standstill has been reached.

NOTICE



No position control of the composite axes during quick stop

During the quick stop deceleration, there is no position control and therefore no coordinated movement of the composite axes according to their definition as a cam or electronic gearing. Each composite axis comes to a standstill at a different time depending on its current speed and the setting for the quick stop deceleration.

► Only use the command in emergencies!

Parameters

The function has no parameters.

Syntax

```
tecGroup.Power.QuickStop();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	–	Disabled
Enabled	√	Disabled
Running	√	Disabled
Suspended	√	Disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

The state transition cannot be processed within one MCX cycle. Therefore, before calling further commands for the group, query in the program whether the new state has been reached, as follows:

```
When tecGroup.State.IsDisabled Continue;
```

82.3 Example task MCTechno.Power

Here you will find an example task for a single axis that contains various Motion API functions:

```
Task Sample1 Autorun
    //Enable the composite axes
    If tecGroup.State.IsDisabled Then
        tecGroup.Power.Enable();
    End_if;
    When tecGroup.State.IsEnabled Continue;

    //a process with the group can be placed here, or other tasks move the
group
    //…

    // Disable the group
    If tecGroup.State.IsRunning Then    //if the group is still moving, stop
it with a suitable deceleration

        tecGroup.MoveHalt.Start(Techno.Members.Master,MCTechnoHaltModes.Norm
al,1000.0);
    When tecGroup.Mechanism.Slope.IsStopped Continue;

    End_if;
    tecGroup.Power.Disable(); // Disable the group
    When tecGroup.State.IsDisabled Continue;

End_Task;
```

83 MCTechno.Settings

The **MCTechno.Settings** functionality allows you to adjust or save all parameters of an MCX technology group together.

Do not adjust parameters while saving is in progress

Saving the parameters to a file is a process that is executed in the background, i.e. it extends over several MCX cycles. If parameter values are changed during saving, it is unpredictable whether the old or the new value is written to the file.

How parameter management works

Parameters that are saved in the motion setup of JetSym are stored in a text file on the PC and, on request, also in the file system of the control unit (subfolder Motion). The file name of the parameter file on the control unit is *MCTechno_X.ini*, where *X* stands for the number of the technology group ID. The technology group ID is displayed on the Properties page in the motion setup of the MCX group. In addition, the tooltip in the hardware tree displays the technology group ID when the mouse pointer is placed on the group.

When the control unit boots, the MCX kernel reads the data from the *MCTechno_X.ini* file and uses it for the group concerned. If such a file is present on the control unit, no further parameterization is required. If there is no corresponding file for the group, the parameters are loaded with default values.

Normally, no adjustment to the file from the STX program is required.

In very few exceptional cases, however, it can make sense to read in the parameter file on the control unit again or to save the parameters. When saving, however, bear in mind that the parameter file saved in the JetSym project may then no longer match the parameters effective on the machine.

83.1 Syntax for MCTechno.Settings

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Settings.<function>(optional <parameter>;
```

83.2 Functions of MCTechno.Settings

The functionality provides the following functions:

– SaveToFile()
– LoadFromFile()

83.2.1 SaveToFile()

Parameters

Parameter	Data type	Meaning
Suffix	Integer (optional)	Activates saving of the parameters of the MCX group to a file in the subfolder Motion in the file system of the control unit. Depending on the parameter passed, different files are created or overwritten: – Parameter equal to `0` or no parameter: The parameter file *MCTechno_X.ini* 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. *Y* is equal to the `Suffix` parameter.

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 MCTechno_1_Tmp3.ini
(if tecGroup has Techno ID 1)
```

State management

The function can be called in any state.

83.2.2 LoadFromFile()

Parameters

Parameter	Data type	Description
Suffix	Integer (optional)	Activates the adoption of the parameters of the MCX group from a file in the subfolder Motion in the file system of the control unit. Depending on the parameter passed, different files are read. – Parameter equal to `0` or no parameter: The data from the file *MCTechno_X.ini* is read and used for the group concerned. – Parameter not equal to `0`: The data from the file *MCTechno_X_TmpY.ini* is read and used for the group concerned. *Y* is equal to the `Suffix` parameter.

Syntax

```
tecGroup.Settings.LoadFromFile();    //-> Reads the file MCTechno_1.ini (if
tecGroup has Techno ID 1)
tecGroup.Settings.LoadFromFile(3);    //-> Reads the file
MCTechno_1_Tmp3.ini (if tecGroup has Techno ID 1)
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	X	Enabled
Running	X	Running
Suspended	X	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

84 MCTechno.State

The **MCTechno.State** functionality allows you to query various information about a technology group, e.g. operating mode, motion type etc.

Basic and further information

In this context, also observe the following information:

– For state transitions, section MCTechno.State.Transitions [▶ 498];

– For a detailed description of the structure of the MCX core, the JetSym online help (JetSym Motion and MotionControl > JetSym MotionControl > MotionControl basics > Operating concept > Overview: Operating concept)

– For basic information, Motion Control Application Note No. 54 (MCX motion control basics).

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

Operating state

At any time, a group is in one, and only one, clearly defined operating state (`OperatingState`). If, for example, the group is in the **Running** operating state, the **Enabled** operating state is not active, although of course all composite axes have pulse enable.

In a particular state, always only a subset of all function calls is permitted. The state model for the axis is as follows:

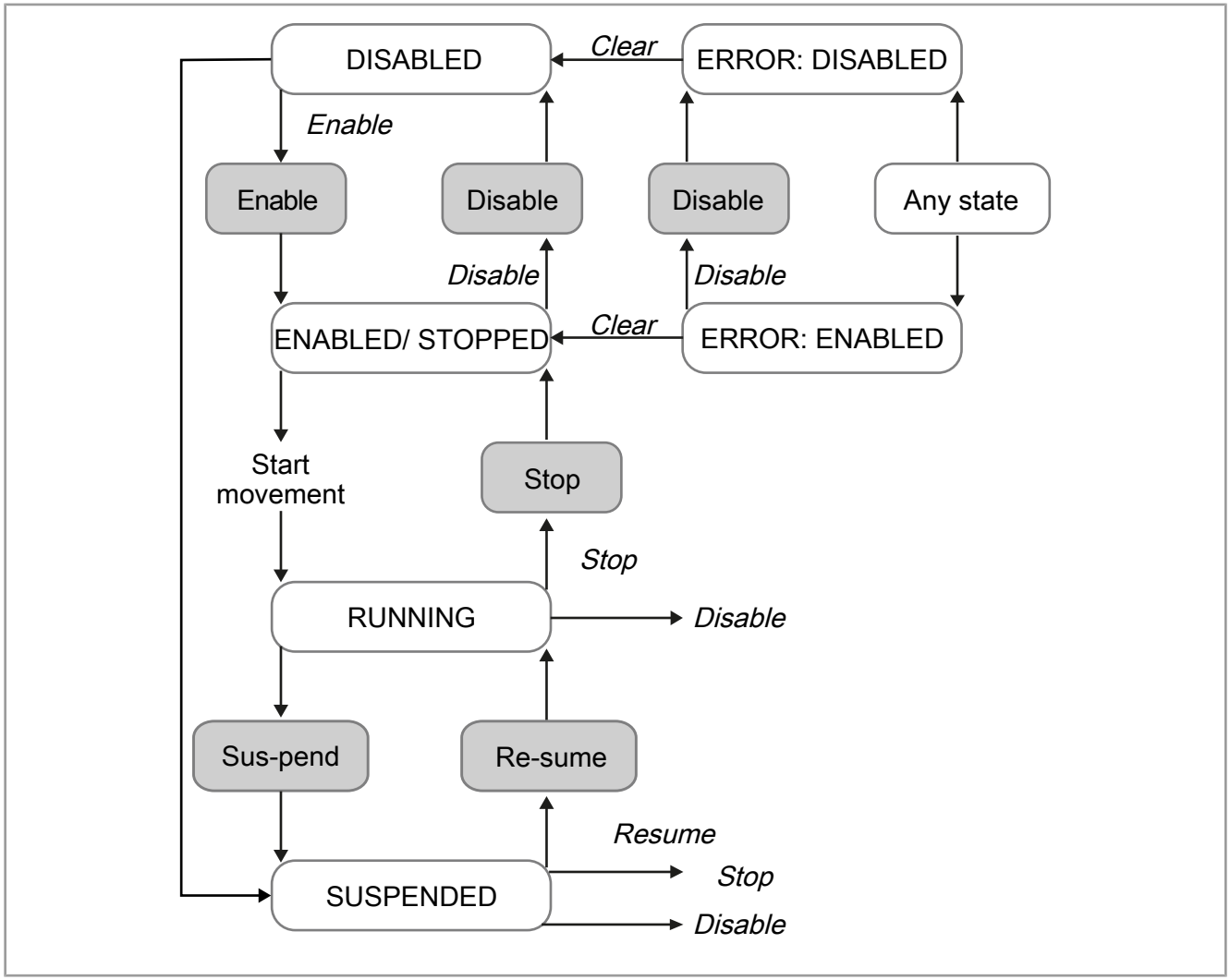

Fig. 15: State model for MCX groups

Operating state	Meaning
Inactive (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
Disabled (disabled)	– The controllers of the servo amplifiers of all composite axes are open (no pulse enable). – No torque is generated.
Enabled (enabled)	– The controllers of all composite axes are closed (pulse enable). – The setpoint interpolator is at standstill.
Running (running)	At least one axis is moving, i.e. the setpoint interpolator passes motion setpoints on to the servo amplifier.
Suspended (suspended)	– Corresponds to the **Enabled** operating state, with the exception that the target position is retained. A movement suspended with `MCTechno.State.Transitions.Suspend()` can be continued with the `MCTechno.State.Transitions.Resume()` command (state transition resume).
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control (error level 4).
DisabledError (Error: disabled)	An error has occurred. After the error response, the control of all axes is disabled (error levels 5 and 6).

Tab. 10: Operating states of technology groups

Operating mode

The operating mode (`OperatingMode`) indicates how the technology group basically works. There are 3 operating modes:

Operating mode	Meaning
`Inactive` (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
`Normal` (normal)	The group can be used. Travel commands are always executed in displacing mode, as buffered mode is not possible with technology groups.
`Error` (error)	An operating error is pending. The operating state is retained, but no new functions and parameters can be passed.

Tab. 11: Operating modes of MCX technology groups

Motion type

The motion type (`TypeOfMotion`) indicates which type of movement the setpoint generator is currently generating for the technology group.

Querying the motion type is useful, for example, in a monitoring task in which the process is not known. Based on the motion type active before switching off, it is possible to control the restart correctly.

Operating state	Meaning
No movement (`MoveNone`)	The setpoint generator outputs standstill because no movement is requested.
Positioning (`MovePtp`)	The master axis of the technology group is currently executing, or last executed, a point-to-point movement. This motion type is also active after the group has been activated for the first time. The motion type of a follower axis that is started independently of the master axis is not taken into account.
Continuous movement (`MoveVelocity`)	The master axis of the technology group is currently executing, or last executed, a continuous movement. The motion type of a follower 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 functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.State.<value access>
```

84.2 Properties of MCTechno.State

The functionality provides the following properties:

- Value
- IsDisabled
- IsEnabled
- IsRunning
- IsSupended
- IsDisabledError
- IsEnabledError
- IsInactive

– OperatingMode
– TypeOfMotion
– Transitions (for a description, see MCTechno.State.Transitions [▶ 498])

84.2.1 Value

Description

Data type	Meaning
OperatingStates	Reads the operating state of the technology group (for a detailed description, see section *Operating state* in MCGeo.State.Transitions [▶ 389]).
OperatingStates.Inactive	Inactive
OperatingStates.Disabled	Disabled
OperatingStates.Enabled	Enabled
OperatingStates.Running	Running
OperatingStates.Suspended	Suspended
OperatingStates.EnabledError	Error: enabled
OperatingStates.DisabledError	Error: disabled

Syntax

```
OpState := tecGroup.State.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.2 IsDisabled

Description

Data type	Meaning
Bool	Queries whether the group is in the **Disabled** operating state.

Syntax

```
When techgroup.State.IsDisabled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.3 IsDisabledError

Description

Data type	Meaning
Bool	Queries whether the group is in the **Error: disabled** operating state.

Syntax

```
When tecGroup.State.IsDisabledError Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.4 IsEnabled

Description

Data type	Meaning
Bool	Queries whether the group is in the **Enabled** operating state.

Syntax

```
When tecGroup.State.IsEnabled Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.5 IsEnabledError

Description

Data type	Meaning
Bool	Queries whether the group is in the **Error: enabled** operating state.

Syntax

```
When tecGroup.State.IsEnabledError Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.6 IsInactive

Description

Data type	Meaning
Bool	Queries whether the group is in the **Inactive** operating state.

Syntax

```
When tecGroup.State.IsInactive Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.7 IsRunning

Description

Data type	Meaning
Bool	Queries whether the group is in the **Running** operating state.

Syntax

```
When tecGroup.State.IsRunning Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.8 IsSuspended

Description

Data type	Meaning
Bool	Queries whether the group is in the **Suspended** operating state.

Syntax

```
When tecGroup.State.IsSuspended Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.9 OperatingMode.Value

Description

Data type	Meaning
Type MCOperatingModes	Reads the operating mode of the group (for a detailed description, see section Operating mode [▶ 389]).
MCOperatingModes.Inactive	The group is inactive, e.g. because it has not been activated.
MCOperatingModes.Normal	The group has been activated and is not in the error state.
MCOperatingModes.Error	The group is in the error state after an operating error.

Syntax

```
OpModeAxis := tecGroup.State.OperatingMode.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInInactiveMode

Description

Data type	Meaning
Bool	Queries whether the axis is in the *Inactive* operating mode.

Syntax

```
When tecGroup.State.OperatingMode.IsInInactiveMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInErrorMode

Description

Data type	Meaning
Bool	Queries whether the group is in the *Error* operating mode.

Syntax

```
When tecGroup.State.OperatingMode.IsInErrorMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

OperatingMode.IsInNormalMode

Description

Data type	Meaning
Bool	Queries whether the group is in the *Normal* operating mode.

Syntax

```
When tecGroup.State.OperatingMode.IsInNormalMode Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

84.2.10 TypeOfMotion.Value

Description

Operating state	Meaning
Type MCTechnoTypesOfMotion	Motion type the technology group is currently in (for a detailed description, see section MCTechno.State [▶ 491]).

Operating state	Meaning
MCTechnoTypesOfMotion.MoveNone	Motion type `None`: The group is not active.
MCGeoTypesOfMotion.MovePtp	The motion type `Positioning` is displayed after the technology group has been activated until a different motion type is executed.
MCGeoTypesOfMotion.MoveVelocity	The motion type `Continuous` is displayed until a different motion type is executed by the technology group.

Syntax

```
TypeOfMotion := tecGroup.State. TypeOfMotion.Value;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveNone

Description

Data type	Meaning
Bool	Queries whether the group is in the motion type `None`.

Syntax

```
When tecGroup.State.TypeOfMotion.IsMoveLinear Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMovePtp

Description

Data type	Meaning
Bool	Queries whether the group is currently executing a `positioning movement`.

Syntax

```
When tecGroup.State.TypeOfMotion.IsMovePtp Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

TypeOfMotion.IsMoveVelocity

Description

Data type	Meaning
Bool	Queries whether the group is currently executing a `continuous movement`.

Syntax

```
When tecGroup.State.TypeOfMotion.IsMoveVelocity Continue;
```

State management

The property is used for diagnostic purposes. It can only be accessed for reading.

The property can be read in any state.

85 MCTechno.State.Transitions

The **MCTechno.State.Transitions** functionality allows you to control state transitions of a technology group for special applications.

The state transitions of a group are usually triggered automatically by the motion commands and the behavior of the axis movement. Direct control of the state management is only required in exceptional cases, e.g. to respond in the event of an error. The **MCTechno.State.Transitions** functionality, as well as `Diagnostics.ClearErrors()`, allow operating state transitions to be triggered explicitly.

The **MCTechno.State** functionality allows you to query various information about a technology group, e.g. operating mode, motion type etc.

Basic and further information

In this context, also observe the following information:

– For a description of the other functionalities of MCTechno.State, see section MCTechno.State [▶ 489];
– For a detailed description of the structure of the MCX core, the JetSym online help (JetSym Motion and MotionControl > JetSym MotionControl > MotionControl basics > Operating concept > Overview: Operating concept)
– For basic information, Motion Control Application Note No. 54 (MCX motion control basics).

ⓘ Info	**Further information**
	For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

Operating state

At any time, a group is in one, and only one, clearly defined operating state (`OperatingState`). If, for example, the group is in the **Running** operating state, the **Enabled** operating state is not active, although of course all composite axes have pulse enable.

In a particular state, always only a subset of all function calls is permitted. The state model for the axis is as follows:

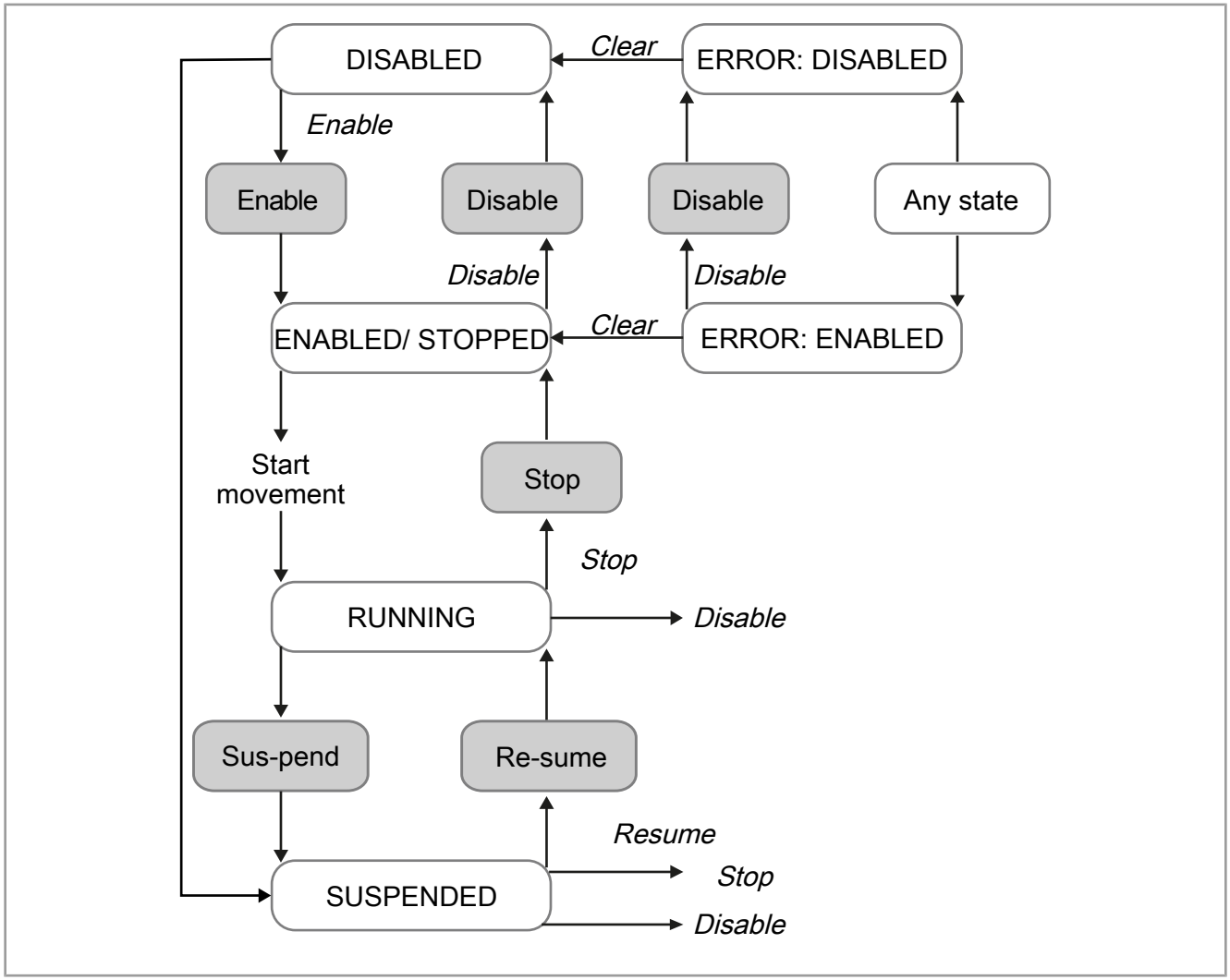

Fig. 16: State model for MCX groups

Operating state	Meaning
Inactive (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
Disabled (disabled)	– The controllers of the servo amplifiers of all composite axes are open (no pulse enable). – No torque is generated.
Enabled (enabled)	– The controllers of all composite axes are closed (pulse enable). – The setpoint interpolator is at standstill.
Running (running)	At least one axis is moving, i.e. the setpoint interpolator passes motion setpoints on to the servo amplifier.
Suspended (suspended)	– Corresponds to the **Enabled** operating state, with the exception that the target position is retained. A movement suspended with `MCTechno.State.Transitions.Suspend()` can be continued with the `MCTechno.State.Transitions.Resume()` command (state transition resume).
EnabledError (Error: enabled)	An error has occurred. After the error response, all axes remain in control (error level 4).
DisabledError (Error: disabled)	An error has occurred. After the error response, the control of all axes is disabled (error levels 5 and 6).

Tab. 12: Operating states of technology groups

Operating mode

The operating mode (`OperatingMode`) indicates how the technology group basically works. There are 3 operating modes:

Operating mode	Meaning
`Inactive` (inactive)	The group cannot accept any commands, e.g. because it has not been activated.
`Normal` (normal)	The group can be used. Travel commands are always executed in displacing mode, as buffered mode is not possible with technology groups.
`Error` (error)	An operating error is pending. The operating state is retained, but no new functions and parameters can be passed.

Tab. 13: Operating modes of MCX technology groups

Motion type

The motion type (`TypeOfMotion`) indicates which type of movement the setpoint generator is currently generating for the technology group.

Querying the motion type is useful, for example, in a monitoring task in which the process is not known. Based on the motion type active before switching off, it is possible to control the restart correctly.

Operating state	Meaning
No movement (`MoveNone`)	The setpoint generator outputs standstill because no movement is requested.
Positioning (`MovePtp`)	The master axis of the technology group is currently executing, or last executed, a point-to-point movement. This motion type is also active after the group has been activated for the first time. The motion type of a follower axis that is started independently of the master axis is not taken into account.
Continuous movement (`MoveVelocity`)	The master axis of the technology group is currently executing, or last executed, a continuous movement. The motion type of a follower 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 functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.State.Transitions.<function>(optional <parameter>);
```

85.2 Functions of MCTechno.State.Transitions

The functionality provides the following functions:

- Enable()
- Disable()
- Stop()
- ErrorDisable()
- ErrorStop()
- ErrorLockUserInput()
- Suspend()
- Resume()

85.2.1 Enable()

The `State.Transistions.Enable()` function is identical to the `MCTechno.Power.Enable()` function. A detailed description of the function can be found in section MCTechno.Power.Enable() [▶ 483].

Syntax

```
tecGroup.State.Transitions.Enable();
```

85.2.2 Disable()

Depending on the parameter passed (see the following table), the function performs the same action as the commands `MCTechno.Power.Disable()` and `MCTechno.Power.Qickstop()`. For details, refer to the descriptions of MCTechno.Power [▶ 483].

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionDisableModes	Mode for executing the `Disable()` function
	MCStateTransitionDisable-Modes.ImmediatePulseBlocking	Performs the same function as the command `MCTechno.Power.Disable()` or `MCTechno MCTechno.Power.Disable(MCPowerDisable-Modes.Forced)`
	MCStateTransitionDisable-Modes.VelocityControlSlope	Performs the same function as the command `MCTechno.Power.QuickStop()`.

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), the function performs the same action as the command MCGeo.Stop() [▶ 400].

The difference is that passing a function parameter is mandatory with `MCGeo.State.Transition.Stop()`, whereas it is optional with `MCGeo.Stop()` (for details, see there).

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionStop-Modes	Two deceleration values are available.
	StopModes.ProgDeceleration	The deceleration last specified for the group is used for stopping.
	StopModes.MaxDeceleration	The maximum acceleration of the group is used as deceleration for stopping. It was either specified during commissioning in the motion setup under "Maximum deceleration" or programmed with the command `geoGroup.Mechanism.Limitations.MaxAcceleration`.

Syntax

```
geoGroup.State.Transitions.Stop(MCStateTransitionStopModes.ProgDecelera-
tion);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√		Enabled
Running	√	Running	Enabled
Suspended	√		Enabled
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

State monitoring

Various information is available for monitoring the positioning.

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis is stopped.

```
When geoGroup.State.IsEnabled Continue; //The group is in position control
and at standstill.
When geoGroup.State.IsRunning Continue; //The setpoint generator of at least
one composite axis is generating a movement.
```

– The **ramp status** indicates which section of the movement 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;
```

– The **target window status** is not affected by this function.

85.2.4 ErrorDisable()

This function triggers an error in the path group that leads to the **Error: disabled** state. The error response can be set (see property `ErrorDisableMode`).

Parameters

The function has no parameters.

Syntax

```
tecGroup.State.Transitions.ErrorDisable();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: disabled
Enabled	√	Error: disabled
Running	√	Error: disabled
Suspended	√	Error: disabled
Error: enabled	√	Error: disabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

85.2.5 ErrorStop()

This function triggers an error in the group that leads to the **Error: enabled** state. The error response can be set (see property ErrorStopMode [▶ 507]).

Parameters

The function has no parameters.

Syntax

```
tecGroup.State.Transitions.ErrorStop();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Error: enabled
Enabled	√	Error: enabled
Running	√	Error: enabled
Suspended	√	Error: enabled
Error: enabled	-	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

85.2.6 ErrorLockUserInput()

This function triggers an operating error in the group.

Parameters

The function has no parameters.

Syntax

```
tecGroup.State.Transitions.ErrorLockUserInput();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	√	Disabled
Enabled	√	Enabled
Running	√	Running
Suspended	√	Suspended
Error: enabled	–	Error: enabled
Error: disabled	–	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

85.2.7 Suspend()

This function suspends a running group movement. The group is stopped either with the last programmed deceleration ramp or with the configured maximum deceleration. The state after stopping is not **Enabled** but **Suspended**. With the `Resume()` function, the suspended movement of a group can be resumed and moved to the original target position (of the master axis).

If the master axis can no longer be stopped before the actual target position with the specified deceleration, the command is ignored and the positioning is completed. The state is then **Enabled** and not **Suspended**.

Parameters

Parameter	Data type	Meaning
Mode	MCStateTransitionStopModes	Mode for the deceleration ramp of the `Suspend()` function
	MCStateTransitionStopModes.ProgDeceleration	Uses the deceleration last programmed for the master axis.
	MCStateTransitionStopModes.MaxDeceleration	Uses the maximum deceleration configured for the master axis.

Syntax

```
tecGroup.State.Transitions.Suspend(MCStateTransitionStopModes.ProgDecelera-
tion);
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	X	Enabled
Running	√	Suspended
Suspended	–	Suspended
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

85.2.8 Resume()

The function continues a positioning that was suspended with the Suspend() [▶ 504] function. The originally specified target position of the master axis is approached. The speed and acceleration values are also adopted from the original positioning.

The `Resume()` function only takes effect if the group is in the **Suspended** state.

Parameters

The function has no parameters.

Syntax

```
tecGroup.State.Transitions.Resume();
```

State management

The following table applies to writing the function:

Initial state	Allowed	Final state
Disabled	X	Disabled
Enabled	X	Enabled
Running	–	Running
Suspended	√	Running
Error: enabled	X	Error: enabled
Error: disabled	X	Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

85.3 Properties of MCTechno.State.Transitions

The functionality provides the following properties:

– ErrorDisableMode
– ErrorStopMode
– ErrorLevel

85.3.1 ErrorDisableMode

Description

Data type	Meaning
MCStateTransitionDisableModes	Error response used when the `ErrorDisable()` function is triggered.
MCStateTransitionDisableModes.Velocity-ControlSlope	Stops the group with the quick stop deceleration. When standstill is reached, the servo amplifier is disabled.
MCStateTransitionDisableModes.Immedi-atePulseBlocking	Immediately disables the servo amplifier for torque generation (pulse inhibit).

Syntax

```
tecGroup.State.Transitions.ErrorDisableMode := MCStateTransitionDisable-
Modes.ImmediatePulseBlocking;
ErrorResponse := tecGroup.State.Transitions.ErrorDisableMode;
```

State management

The property can be read and written in any state.

On read access to the property, 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 triggered.
MCStateTransitionErrorStop-Modes.ProgDeceleration	Stops the group with the programmed deceleration.
MCStateTransitionErrorStop-Modes.MaxDeceleration	Stops the group with the maximum deceleration.
MCStateTransitionErrorStop-Modes.AtBlockEnd	This mode has no meaning for the technology group, as buffered mode is not possible with the technology group.

Syntax

```
tecGroup.State.Transitions.ErrorStopMode := MCStateTransitionStop-
Modes.MaxDeceleration;
ErrorResponse := tecGroup.State.Transitions.ErrorStopMode;
```

State management

The property can be read and written in any state.

85.3.3 ErrorLevel

Description

This property allows you to set the error level that the technology group assumes in the event of a drive error. By default, this is error level 4 (operational stop and **Error: enabled** operating state).

Data type	Meaning
MCStateTransitionErrorLevels	Error level of the path group in the event of a drive error. Detailed information on the differences between the error response levels can be found in the JetSym online help under *Error management*.
MCStateTransitionErrorLevels.Message	The group responds with error level 1: a message/warning is created.
MCStateTransitionErrorLevels.Channel-Stop	The group responds with error level 2: the command channel is locked.
MCStateTransitionErrorLevels.Cannel-StopClearBuffer	Exception for technology groups: Identical function to error response level 2, as buffered mode is not possible with technology groups.
MCStateTransitionErrorLevels.Stop	The group responds with error level 4: an operational stop is executed (default).
MCStateTransitionErrorLevels.Quickstop (default)	The group responds with error level 5: an emergency stop is executed.
MCStateTransitionErrorLevels.PulsBlock-ing	The group responds with error level 6: the controller is disabled (pulse inhibit).

Syntax

```
tecGroup.State.Transitions.ErrorLevel := MCStateTransitionErrorLevels.Puls-
Blocking;
ResponseOnDriveError := tecGroup.State.Transitions.ErrorLevel;
```

State management

The property can be read and written in any state.

86 MCTechno.Stop

The **MCTechno.Stop** functionality allows you to stop a moving technology group with a defined ramp.

Unlike the `MoveHalt.Start()` function, the `Stop` function always affects the entire group.

The functionality has no further functions or properties.

ⓘ Info

Further information

For procedures and sample projects on working with groups, please refer to **Motion Control Application Notes (APN) Nos. 49 - 56**. These are available for download on the Bucher Automation website.

86.1 Syntax for MCTechno.Stop

The easiest way to enter the functionality in the JetSym editor is via IntelliSense, as follows:

```
<group name>.Stop(optional <parameter 1>)
```

86.2 Functions of MCTechno.Stop

`MCTechno.Stop` is a function of the group object.

86.2.1 Stop()

Parameters

Observe the following behavior:

– The function executes the stop command immediately. The parameter passed (see table) specifies the maximum deceleration for this.

– If no parameter is passed, the `StopMode` passed last is used.

– When stopping, the follower axes are brought to a standstill synchronously with the master axis according to their coupling function.

Parameter	Data type	Meaning
StopModes	Enum StopModes (optional)	Two deceleration values are available.
	StopModes.ProgDeceleration	The deceleration last specified for the group is used for stopping (default).
	StopModes.MaxDeceleration	The maximum acceleration of the master axis is used as deceleration for stopping. It was either specified during commissioning in the motion setup under Maximum deceleration or programmed with the command `axMasterAxis.Mechanism.Limitations.MaxAcceleration`.

Syntax

```
tecGroup.Stop(StopModes.ProgDeceleration);
tecGroup.Stop(StopModes.MaxDeceleration);
```

State management

The following table applies to writing the function:

Initial state	Allowed	During execution	Final state
Disabled	X		Disabled
Enabled	√		Enabled
Running	√	Running	Enabled
Suspended	√		Enabled
Error: enabled	X		Error: enabled
Error: disabled	X		Error: disabled

Where:

√ the state in which the function can be called.

– the state in which calling the function is not recommended but does not trigger an error.

X the state in which calling the function is not allowed and triggers an error.

State monitoring

Various information is available for monitoring the `Stop()` function.

– The **state of the axis** indicates whether the setpoint generator is generating a movement or the axis is stopped.

```
When tecGroup.State.IsEnabled Continue; //The group is in position control
and all axes are at standstill.
When tecGroup.State.IsRunning Continue; //The setpoint generator of at least
one composite axis is generating a movement.
```

– The **ramp status** indicates which section of the movement 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;
```

– The **target window status** is not affected by this function.

Glossary

Error
Notification type indicating that an error threshold has been reached. Errors trigger a system response and must be acknowledged in order to continue operation. See also warnings and messages.

Master axis
In connection with the MCCouplingGroup functionality, the master axis is a superimposed axis that must not be confused with the master axis of a technology group, as the functions are fundamentally different.

Message
Notification type of the motion control system that contains important information or warns that an error threshold is about to be reached. Messages do not trigger a system response, but must be acknowledged. See also warnings.

Synchronous axis
Axis that is moved in sync with the path movement without being mechanically part of the path movement.

Technology group
A technology group is a coupling of several MCX axes in which one or more follower axes are moved depending on the position of a master axis.

Warning
Notification type of EtherCAT devices that warns that an error threshold is about to be reached. Warnings do not trigger a system response and are cleared automatically as soon as the cause of the warning no longer exists. See also messages.

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

www.bucherautomation.com