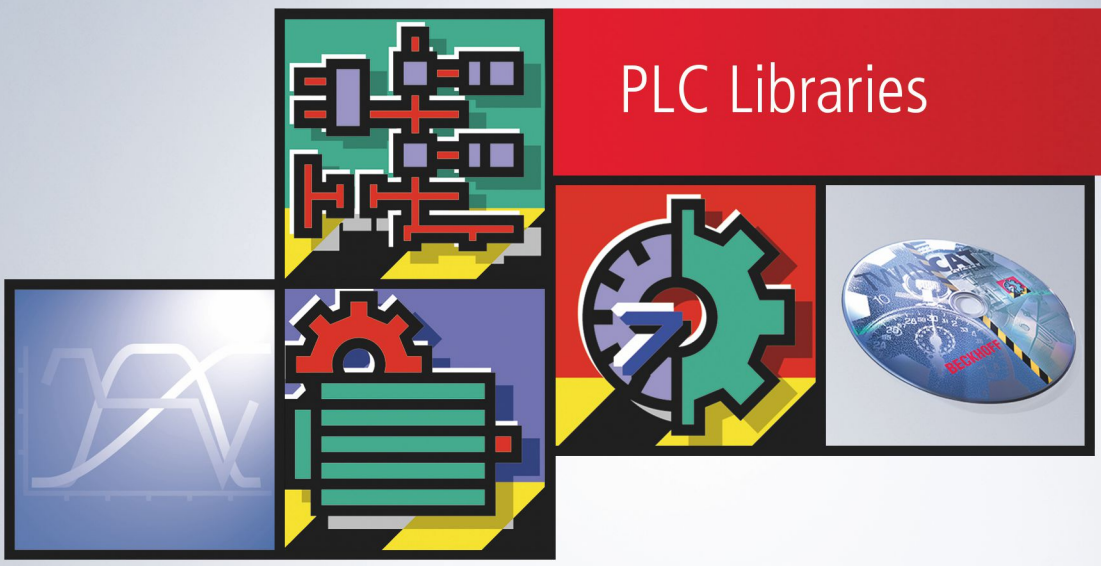
BECKHOFF New Automation Technology

Handbuch | DE

TX1200

TwinCAT 2 | PLC-Bibliothek: TcMC2



20.07.2023 | Version: 1.3

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Die TwinCAT Motion Control SPS-Bibliothek TcMC2 enthält Funktionsbausteine zur Programmierung von Maschinenapplikationen und ist eine Weiterentwicklung der TcMC. Die TcMC2 richtet sich nach der ebenfalls überarbeiteten PLCopen Spezifikation für Motion Control Funktionsbausteine V2.0 (www.PLCopen.org).

Kompatibilität

Die Motion Control Bibliothek TcMC2 führt weiterentwickelte und auch neue Funktionen ein. Die Funktionsbausteine erfüllen besser die Anforderungen der PLCopen Spezifikation und sind dadurch nicht zur ersten Version TCMC kompatibel. Anwendern, die bestehende Projekte pflegen, wird empfohlen, in diesen Projekten mit der klassischen TcMC weiterzuarbeiten. Die TcMC2 sollte für neue Projekte oder bei der Überarbeitung bestehender Projekte zum Einsatz kommen.

Wesentliche Neuerungen

Ein wichtiges neues Feature der TcMC2 gegenüber der TcMC ist der sogenannte Buffer-Mode. Der Buffer-Mode erlaubt es, Move-Kommandos zu puffern und somit einen kontinuierlichen Positionierablauf ohne Zwischenstopp zu erreichen. Zwei Bewegungskommandos können an einer bestimmten Position mit definierter Geschwindigkeit ineinander übergehen.

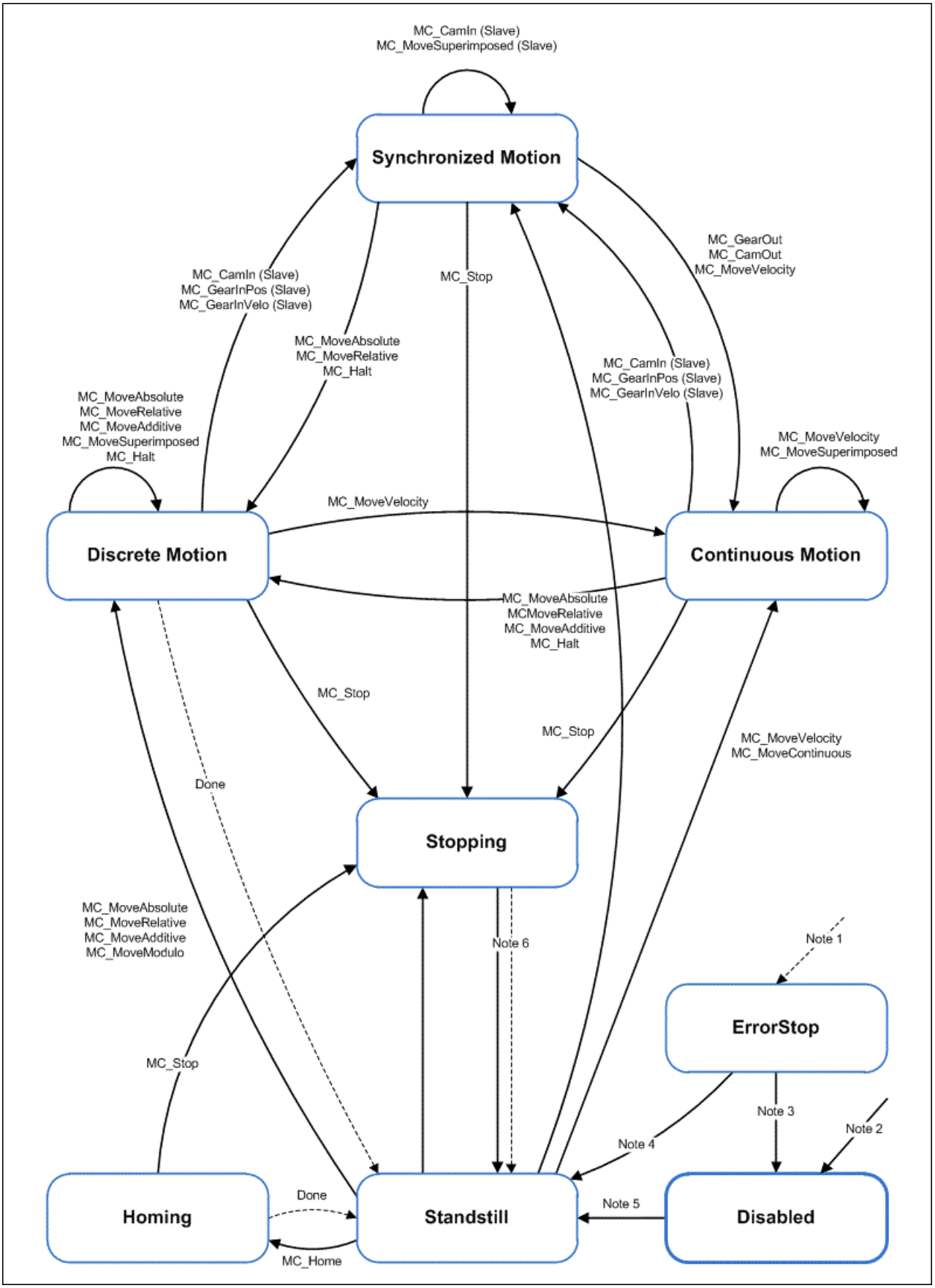
Move-Kommandos können durch weitere Move-Kommandos noch während der Ausführung abgelöst werden. Es ist dadurch wesentlich einfacher geworden, während der Bewegung die Zielposition oder die Fahrgeschwindigkeit anzupassen.

TwinCAT Version

Die Bibliothek TcMC2 ist ab der TwinCAT Version 2.10 Build 1340 einsetzbar. Bei remote programmierten Steuerungen ist darauf zu achten, dass sowohl auf dem Programmier-PC, als auch auf dem Steuerungs-PC eine entsprechende Version installiert ist. Bei Steuerungen mit dem Betriebssystem Windows CE ist die Version des installierten Images ausschlaggebend. Hier wird ein Windows CE Image ab der Version 3.08 benötigt.

3 Das Zustandsdiagramm

Das folgende Zustandsdiagramm definiert das Verhalten einer Achse für den Fall, dass mehrere Funktionsbausteine für diese Achse gleichzeitig aktiv sind. Die Kombination mehrerer Funktionsbausteine ist nützlich, um komplexere Bewegungsprofile zu erzeugen oder um Ausnahmezustände in einem Programmablauf zu behandeln.



Note 1	Aus irgendeinem Zustand, in dem ein Fehler auftritt
Note 2	Aus irgendeinem Zustand, wenn *MC_Power.Enable*= FALSE und die Achse hat keinen Fehler

Note 3	MC_Reset und *MC_Power.*Status = FALSE
Note 4	MC_Reset und *MC_Power.*Status = TRUE und *MC_Power*.Enable = TRUE
Note 5	MC_Power.Status = TRUE und *MC_Power*.Enable = TRUE
Note 6	MC_Stop.Done= TRUE und *MC_Stop.Execute* = FALSE

Als Grundregel werden Bewegungskommandos grundsätzlich sequentiell abgearbeitet. Alle Kommandos arbeiten in diesem Achs-Zustandsdiagramm.

Die Achse befindet sich immer in einem der definierten Zustände. Jedes Bewegungskommando, das eine Transition verursacht, ändert den Zustand der Achse und ändert als Folge daraus, das Bewegungsprofil. Das Zustandsdiagramm ist eine Abstraktionsebene, die den realen Zustand der Achse, vergleichbar zum Prozessabbild von E/A-Punkten, wiederspiegelt. Der Zustand der Achse wechselt sofort mit dem auslösenden Kommando.

Das Zustandsdiagramm zielt zunächst auf Einzelachsen ab. Multi-Achsbausteine, wie *MC_CamIn* oder *MC_GearIn* beeinflussen die Zustände mehrerer Achsen, die jedoch immer auf einen Einzelachszustand der beteiligten Achsen zurückgeführt werden können. Beispielsweise kann ein Kurvenscheiben-Master im Zustand *Continous Motion* und der zugehörige Slave im Zustand *Synchronized Motion* sein. Das Ankoppeln eines Slaves hat keinen Einfluss auf den Zustand des Masters.

Der Zustand *Disabled* beschreibt den Grundzustand einer Achse. In diesem Zustand kann die Achse durch keinen Funktionsbaustein bewegt werden. Nachdem der MC_Power [▶ 21] Baustein mit Enable=TRUE aufgerufen wird, wechselt die Achse in den Zustand *Standstill* oder im Fehlerfall in den Zustand *ErrorStop*. Wenn *MC_Power* mit Enable=FALSE aufgerufen wird, wechselt der Zustand nach *Disabled*

Der Zweck des Zustands *ErrorStop* ist die Achse zu stoppen und anschließend keine weiteren Kommandos anzunehmen, bis ein Reset ausgelöst wurde. Der Zustandsübergang Error bezieht sich nur auf tatsächliche Achs-Fehler und nicht auf Ausführungsfehler eines Funktionsbausteins. Achs-Fehler können aber auch am Fehlerausgang eines Funktionsbausteins angezeigt werden.

Funktionsbausteine, die im Zustandsdiagramm nicht aufgeführt werden, beeinflussen den Zustand der Achse nicht. (*MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_ReadActualPosition* und *MC_CamTableSelect.*)

Der Zustand *Stopping* zeigt an, dass sich die Achse in einer Stopprampe befindet. Der Zustand wechselt nach dem vollständigen Stopp nach *StandStill*.

Bewegungskommandos wie *MC_MoveAbsolute*, die aus dem Zustand *Synchronized Motion* herausführen, sind nur dann möglich, wenn sie in den Achs-Parametern explizit erlaubt werden. Abkoppelkommandos wie *MC_GearOut* sind unabhängig davon möglich.

4 Allgemeine Regeln für MC-Funktionsbausteine

Für alle MC-Funktionsbausteine gelten die folgenden Regeln, die eine definierte Abarbeitung durch das SPS-Programm sicherstellen.

Ausschließlichkeit der Ausgänge

Die Ausgänge *Busy, Done, Error* und *CommandAborted* schließen sich gegenseitig aus, es kann also an einem Funktionsbaustein nur einer dieser Ausgänge zu einer Zeit TRUE sein. Sobald der Eingang *Execute* TRUE wird, muss einer der Ausgänge TRUE werden. Ebenfalls kann nur einer der Ausgänge *Active, Error, Done* und *CommandAborted* zu einer Zeit TRUE sein.

Eine Ausnahme von dieser Regel ist MC_Stop [▶ 83]. *MC_Stop* setzt *Done*=TRUE sobald die Achse gestoppt ist. Dennoch bleiben *Busy* und *Active* TRUE, da die Achse verriegelt wird. Erst nachdem *Execute*=FALSE gesetzt wird, wird die Achse entriegelt und *Busy* sowie *Active* auf FALSE gesetzt.

Ausgangs-Zustand

Die Ausgänge *Done, InGear, InSync, InVelocity, Error, ErrorID* und *CommandAborted* werden mit einer fallenden Flanke am Eingang *Execute* zurückgesetzt, wenn der Funktionsbaustein nicht aktiv ist (Busy=FALSE). Allerdings beeinflusst eine fallende Flanke an *Execute* die Kommandoausführung nicht. Falls *Execute* bereits während der Kommandoausführung zurückgesetzt wird, so ist sichergestellt, dass einer der Ausgänge am Ende des Kommandos für einen SPS-Zyklus gesetzt wird. Erst danach werden die Ausgänge zurückgesetzt.

Falls Execute während der Ausführung eines Kommandos mehrfach getriggert wird, so gibt der Funktionsbaustein keinerlei Rückmeldung, führt aber auch kein weiteres Kommando aus.

Eingangs-Parameter

Die Eingangs-Parameter werden mit steigender Flanke an Execute übernommen. Um die Parameter zu ändern, muss das Kommando neu getriggert werden nachdem es beendet wurde oder es muss während der Kommandoausführung eine zweite Instanz des Funktionsbausteins mit neuen Parametern getriggert werden.

Falls ein Eingangs-Parameter nicht an den Funktionsbaustein übergeben wird, so bleibt der zuletzt an diesen Baustein übergebene Wert gültig. Beim ersten Aufruf ist ein sinnvoller Default-Wert gültig.

Position und Distanz

Der Eingang *Position* bezeichnet einen definierten Wert innerhalb eines Koordinatensystems. *Distance* ist dagegen ein relatives Maß, also der Abstand zweier Positionen. Sowohl *Position*, als auch *Distance* werden in technischen Einheiten, z. B. [mm] oder [°], entsprechend der Skalierung der Achse angegeben.

Dynamik-Parameter

Die Dynamikparameter für Move-Funktionen werden in technischen Einheiten mit der Zeitbasis Sekunde angegeben. Ist eine Achse beispielsweise in Millimetern skaliert, so sind die Einheiten für *Velocity* [mm/s], *Acceleration* [mm/s^2], *Deceleration* [mm/s^2] und *Jerk* [mm/s^3].

Fehlerbehandlung

Alle Funktionsbausteine haben zwei Fehlerausgänge um Fehler während der Kommandoausführung anzuzeigen. *Error* zeigt den Fehler an und *ErrorID* gibt eine ergänzende Fehlernummer aus. Die Ausgänge *Done, InVelocity, InGear* und *InSync* bezeichnen eine erfolgreiche Kommandoausführung und werden nicht gesetzt, wenn *Error* TRUE wird.

Am Ausgang der Funktionsbausteine werden Fehler unterschiedlichen Typs signalisiert. Der Fehlertyp wird nicht explizit angegeben, sondern hängt von der Fehlernummer ab, die systemweit eindeutig vergeben ist.

Fehlertypen

- Funktionsbausteinfehler sind Fehler, die ausschließlich den Funktionsbaustein und nicht die Achse betreffen (z. B. fehlerhafte Parametrierung). Funktionsbausteinfehler müssen nicht explizit zurückgesetzt werden, sondern werden selbständig zurückgesetzt, wenn der Eingang *Execute* zurückgesetzt wird.
- Kommunikationsfehler (der Funktionsbaustein kann die Achse z. B. nicht adressieren). Kommunikationsfehler deuten oft auf eine fehlerhafte Konfiguration oder Parametrierung hin. Ein Reset ist nicht möglich, sondern der Funktionsbaustein kann neu getriggert werden, nachdem die Konfiguration korrigiert wurde.
- Achsfehler (logische NC-Achse) treten üblicherweise während der Fahrt auf (z. B. Schleppabstandsfehler) und führen dazu, dass die Achse in den Fehlerzustand geht. Ein Achsfehler muss durch MC_Reset [▶ 22] zurückgesetzt werden.
- Antriebsfehler (Regelgerät) ziehen evtl. einen Achsfehler, also ein Fehler der logischen NC-Achse, nach sich. In vielen Fällen können Achsfehler und Antriebsfehler gemeinsam durch MC_Reset [▶ 22] zurückgesetzt werden. Abhängig vom Antriebsregler kann aber auch ein separater Reset-Mechanismus notwendig sein (z. B. Schalten einer Reset-Leitung zum Regelgerät).

Verhalten des *Done* Ausgangs

Der *Done* Ausgang (oder auch alternativ *InVelocity, InGear, InSync* etc.) wird gesetzt, wenn ein Kommando erfolgreich ausgeführt wurde. Wenn mit mehreren Funktionsbausteinen an einer Achse gearbeitet wird und das laufende Kommando durch einen weiteren Baustein unterbrochen wird, so wird der *Done* Ausgang des ersten Bausteins nicht gesetzt.

Verhalten des *CommandAborted* Ausgangs

CommandAborted wird gesetzt, wenn ein Kommando durch einen anderen Baustein unterbrochen wird.

Verhalten des *Busy* Ausgangs

Der *Busy* Ausgang zeigt an, dass der Funktionsbaustein aktiv ist. Der Baustein kann nur dann mit einer steigenden Flanke an *Execute* getriggert werden, wenn *Busy* FALSE ist. *Busy* wird sofort mit der steigenden Flanke an *Execute* gesetzt und wird erst zurückgesetzt, wenn das Kommando erfolgreich oder auch nicht erfolgreich beendet wurde. Solange *Busy* TRUE ist, muss der Funktionsbaustein zyklisch aufgerufen werden um das Kommando ausführen zu können.

Verhalten des *Active* Ausgangs

Wenn die Bewegung einer Achse durch mehrere Funktionsbausteine gesteuert wird, so zeigt der Active Ausgang jedes Bausteins an, dass das Kommando durch die Achse ausgeführt wird. Der Zustand Busy=TRUE und Active=FALSE bedeutet, dass das Kommando noch nicht oder nicht mehr ausgeführt wird.

Enable Eingang und *Valid* Ausgang

Im Gegensatz zu *Execute* führt der *Enable* Eingang dazu dass eine Aktion permanent und wiederholt ausgeführt wird, solange *Enable* TRUE ist. MC_ReadStatus [▶ 32] aktualisiert beispielsweise zyklisch den Zustand einer Achse solange *Enable* TRUE ist. Ein Funktionsbaustein mit einem *Enable* Eingang zeigt durch den *Valid* Ausgang an, dass die an den Ausgängen angezeigten Daten gültig sind. Die Daten können jedoch ständig aktualisiert werden während *Valid* TRUE ist.

BufferMode

Einige Funktionsbausteine haben einen Eingang *BufferMode*, über den der Kommandofluss mit mehreren Funktionsbausteinen geregelt wird. Der *BufferMode* kann beispielsweise festlegen, dass ein Kommando ein anderes unterbricht (ungepufferter Betrieb) oder aber dass das nachfolgende Kommando erst im Anschluss an das vorherige Kommando ausgeführt wird (gepufferter Betrieb). Im gepufferten Betrieb kann über den *BufferMode* ebenfalls festgelegt werden, wie die Bewegung am Übergang von einem Kommando zum nächsten aussehen soll. Man spricht hier von einem *Blending*, das die Geschwindigkeit am Übergangspunkt festlegt.

Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.

Im ungepufferten Betrieb führt ein nachfolgendes Kommando zu einem Abbruch eines laufenden Kommandos. Das vorherige Kommando setzt dann den *CommandAborted* Ausgang. Im gepufferten Betrieb wartet ein nachfolgendes Kommando, bis ein laufendes Kommando beendet wurde. Hier ist zu beachten, dass eine Endlosbewegung (MC_MoveVelocity) kein gepuffertes Folgekommando zulässt. Gepufferte Kommandos führen immer sofort zum Abbruch einer Endlosbewegung wie im ungepufferten Betrieb.

Es wird nur ein Kommando gepuffert, während ein anderes ausgeführt wird. Wird mehr als ein Kommando während eines laufenden Kommandos getriggert, so wird das zuletzt gestartete und zu puffernde Kommando mit einem Fehler abgewiesen (Fehler 0x4292 Buffer Full). Falls jedoch das letzte Kommando ungepuffert gestartet wird (*Aborting*), so wird es in jedem Fall aktiv und unterbricht das laufende und ein bereits gepuffertes Kommando.

BufferModes

- Aborting : Default-Modus ohne Pufferung. Das Kommando wird sofort ausgeführt und unterbricht gegebenenfalls ein laufendes Kommando.
- Buffered : Das Kommando wird ausgeführt, sobald die Achse kein anderes Kommando mehr ausführt. Die vorherige Bewegung wird bis zum Stopp ausgeführt und das nachfolgende Kommando wir aus dem Stillstand gestartet.
- BlendingLow : Das Kommando wird ausgeführt, sobald die Achse kein anderes Kommando mehr ausführt. Im Gegensatz zu *Buffered* stoppt die Achse an der vorherigen Zielposition nicht, sondern durchläuft diese Position mit der niedrigeren Geschwindigkeit zweier Kommandos.
- BlendingHigh Das Kommando wird ausgeführt, sobald die Achse kein anderes Kommando mehr ausführt. Im Gegensatz zu *Buffered* stoppt die Achse an der vorherigen Zielposition nicht, sondern durchläuft diese Position mit der höheren Geschwindigkeit zweier Kommandos.
- BlendingNext : Das Kommando wird ausgeführt, sobald die Achse kein anderes Kommando mehr ausführt. Im Gegensatz zu *Buffered* stoppt die Achse an der vorherigen Zielposition nicht, sondern durchläuft diese Position mit der Geschwindigkeit des zuletzt beauftragten Kommandos.
- BlendingPrevious: Das Kommando wird ausgeführt, sobald die Achse kein anderes Kommando mehr ausführt. Im Gegensatz zu *Buffered* stoppt die Achse an der vorherigen Zielposition nicht, sondern durchläuft diese Position mit der Geschwindigkeit des zuerst beauftragten Kommandos.

Grafische Darstellung der BufferModes [▶ 107]

Optionale Blending-Position

Das Blending bei den verschiedenen Buffer-Modes wird jeweils an der Zielposition des gerade laufenden Kommandos durchgeführt. Im Falle des *MoveVelocity* ist keine Zielposition definiert und in anderen Fällen kann es sinnvoll sein, die Blending-Position zu ändern. Dazu kann über den *Options*-Eingang des Funktionsbausteins (siehe unten) eine *BlendingPosition* festgelegt werden, die dann für das neue Kommando verwendet wird. Die optionale *BlendingPosition* muss vor der Zielposition des vorherigen Kommandos liegen, anderenfalls wird das neue Kommando mit einer Fehlermeldung abgewiesen (0x4296). Wenn die optionale *BlendingPosition* bereits überfahren worden ist, so wird das neue Kommando instantan umgesetzt, verhält sich also wie ein *Aborting*-Kommando.

***Options* Eingang**

Viele Funktionsbausteine haben einen *Options* Eingang der in einer Datenstruktur zusätzliche, selten benötigte Optionen enthält. Um die Grundfunktion des Funktionsbausteins auszuführen, werden die Optionen oft nicht benötigt, sodass der Eingang offen bleiben kann. Nur wenn in der Dokumentation ausdrücklich auf bestimmte Optionen hingewiesen wird, muss die *Options* Datenstruktur vom Anwender belegt werden.

Slave-Achsen

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_MoveAbsolute* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.

5 Migration von der TcMC zur TcMC2

Um abzuschätzen, mit welchem Aufwand ein bestehendes Projekt von der Motion Control Bibliothek TcMC auf die erweiterte Bibliothek TcMC2 umzustellen ist, werden hier die wichtigsten Unterschiede und Anpassungen angeführt.

Achsdatenstruktur

Bisher wurden für eine Achse zwei Datenstrukturen angelegt, über die zyklische Daten mit der NC ausgetauscht werden.

NcToPlc_Axis1 AT %I* : NCTOPLC_AXLESTRUCT;

PlcToNc_Axis1 AT %Q* : PLCTONC_AXLESTRUCT;

An die meisten Funktionsbausteine, wie z. B. an den MC_MoveAbsolute [▶ 60] wurde am Eingang *Axis* die Datenstruktur NCTOPLC_AXLESTRUCT übergeben. Einige wenige Funktionsbausteine, wie z. B. der MC_Power [▶ 21], haben zusätzlich die Struktur PLCTONC_AXLESTRUCT erwartet.

In der TcMC2 Umgebung ist die Achsstruktur erweitert worden und es sind alle notwendigen Informationen in nur einer Struktur enthalten, die an jeden MC-Funktionsbaustein übergeben wird.

Axis1 : AXIS_REF [▶ 105];

Die Struktur enthält sowohl die zyklischen Eingangs- und Ausgangsdaten zur NC, als auch zusätzliche Statusinformation. Ein bestehendes Projekt greift üblicherweise auf den Inhalt der NcToPlc-Struktur zu. Die enthaltenen Informationen sind in der Struktur *Axis1* ebenfalls verfügbar und das Anwenderprogramm kann damit angepasst werden.

Beispiel:

TcMC : NcToPlc_Axis1.fPosSoll

TcMC2 : Axis1.NcToPlc.SetPos

Zu beachten ist, dass die Unterelemente der NcToPlc- bzw. PlcToNc-Strukturen jetzt englische Namen enthalten um dem internationalen Markt gerecht zu werden. Die aktuelle Sollposition einer Achse ist beispielsweise nicht mehr mit *fPosSoll*, sondern mit *SetPos* benannt.

Funktionsbausteine

Die Eingangs- und Ausgangsbeschaltung der Funktionsbausteine hat sich gegenüber der TcMC etwas geändert. Im Wesentlichen ist bei Move-Bausteinen die Unterstützung des MC_BufferMode [▶ 107] hinzugekommen. Weiterhin unterstützten die Bausteine jetzt einen Busy und Active-Ausgang. Üblicherweise ist für diese Änderungen nur ein geringer Migrationsaufwand notwendig. Die folgende Tabelle enthält eine Auflistung der Bausteine mit umfangreicheren Änderungen.

TcMC	TcMC2	Anmerkung
MC_GearInFloat	MC_GearIn [▶ 97]	MC_GearIn akzeptiert jetzt den Getriebefaktor als Fließkommawert
MC_NewPos MC_NewPosAndVelo	MC_Move...	Durch den neu eingeführten *BufferMode* ist es mit jedem Move-Baustein möglich der Achse ein neues Fahrziel zuzuweisen oder die Geschwindigkeit zu ändern. Die NewPos Funktionsbausteine werden daher nicht mehr benötigt.
MC_MoveAbsoluteOrRestart	MC_Move...	MoveAbsoluteOrRestart kann durch zwei Instanzen eines Move-Bausteins ersetzt werden (siehe *BufferMode*).

MC_CamIn MC_CamInExt	MC_CamIn	Der neue Funktionsbaustein MC_CamIn führt die Funktion des erweiterten Bausteins MC_CamInExt aus. Die Eingangsbeschaltung wurde entsprechend angepasst.
MC_SetReferenceFlag	MC_Home [▶ 92]	Das Setzen und Zurücksetzen des Referenz-Flags (Achse ist referenziert) kann mit dem Baustein MC_Home erreicht werden.
MC_SetPositionOnTheFly	MC_SetPosition [▶ 23]	Für das fliegende Istwertsetzen wird MC_SetPosition im relativen Modus (Mode=TRUE) verwendet.
MC_SetActualPosition	MC_SetPosition [▶ 23]	MC_SetActualPosition wird durch MC_SetPosition ersetzt. Der neue Funktionsbaustein setzt Ist- und Sollposition.
MC_GearOutExt	MC_Move...	Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist (ab TwinCAT 2.11). Ein Bewegungskommando wie *MC_MoveAbsolute* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.
MC_OrientedStop	MC_MoveModulo [▶ 66]	MC_MoveModulo kann aus dem Stillstand oder aus der Bewegung gestartet werden. Im zweiten Fall verhält sich der Baustein wie *MC_OrientedStop*
MC_Stop	MC_Halt [▶ 81] , MC_Stop [▶ 83]	MC_Halt führt einen normalen Halt während der Bewegung aus. Im Gegensatz dazu verriegelt *MC_Stop* die Achse gegen weitere Bewegungskommandos und sollte nur in besonderen Situationen angewendet werden.
MC_Home	MC_Home [▶ 92]	MC_Home überträgt das *bCalibrationCam* Signal des Homing-Sensors nur noch während der Baustein aktiv ist. Falls ein Homing aus dem SystemManager mit F9 durchgeführt werden soll, muss das Signal an anderer Stelle zur NC übertragen werden. Z. B. durch direkte Zuweisung: *Axis.PlcToNc.ControlDword.5 := HomingSensor;*

Bibliothek TcNC

Die bisherige Bibliothek TcMC benötigte Deklarationen und Funktionen aus der Bibliothek TcNC, sodass diese immer mit in ein Projekt eingebunden wurde. Die neue Bibliothek TcMC2 hat diese Abhängigkeit nicht mehr. Alle notwendigen Deklarationen und Funktionen sind jetzt in der TcMC2 selbst enthalten, sodass die TcNC nicht mehr benötigt wird. Dennoch ist es aus Kompatibilitätsgründen möglich, die TcNC zu verwenden.

Statusinformationen

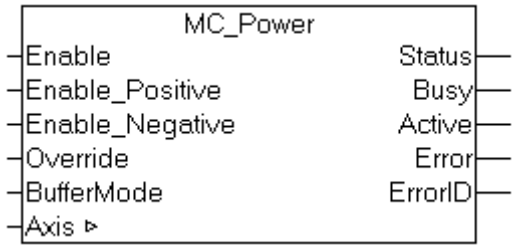
In bisherigen Motion-Anwendungen wurden Statusinformationen einer Achse oft über einen Funktionsaufruf ermittelt (*AxisHasJob(), AxisIsMoving()* etc.). Diese Funktionen können zwar weiterverwendet werden wenn die Bibliothek TcNC eingebunden wird, es wird aber ein anderer Weg empfohlen:

Die Statusinformation einer Achse ist vollständig in der oben genannten Achsdatenstruktur *Axis1*:AXIS_REF [▶ 105] enthalten. Es ist aber notwendig, diese Daten zyklisch durch einen Aufruf des Funktionsbausteins *MC_ReadStatus* oder durch einen Aufruf einer Aktion *Axis1.ReadStatus* am Anfang des SPS-Zyklus zu aktualisieren. Die Statusinformation ist damit an jeder Stelle des Programms zugreifbar und während des SPS-Zyklus aktuell.

6 Organisationsbausteine

6.1 Achsfunktionen

6.1.1 MC_Power



MC_Power schaltet die Software-Freigabe einer Achse. Die Freigabe kann für beide oder nur für eine bestimmte Fahrtrichtung zugeschaltet werden. Am *Status*-Ausgang wird die Betriebsbereitschaft der Achse signalisiert.

Ein Geschwindigkeits-Override beeinflusst prozentual die Geschwindigkeit aller Fahrkommandos.

Zusätzlich zur Software-Freigabe kann es notwendig sein, ein Hardware-Freigabesignal zu schalten um einen Antrieb freizugeben. Dieses Signal wird nicht durch MC_Power beeinflusst und muss durch die SPS separat geschaltet werden.

Abhängig vom Antriebstypen, signalisiert Status auch die Betriebsbereitschaft des Antriebs. Insbesondere digitale Antriebe melden die Betriebsbereitschaft zurück, wogegen analog angeschlossene Antriebe ihre Betriebsbereitschaft nicht zurückmelden können. Im letzen Fall signalisiert Status nur die steuerungsseitige Betriebsbereitschaft.

Eingänge

```
VAR_INPUT
Enable          : BOOL; (* B *)
Enable_Positive : BOOL; (* E *)
Enable_Negative : BOOL; (* E *)
Override        : LREAL (* V *) := 100.0; (* in percent - Beckhoff proprietary input *)
BufferMode      : MC_BufferMode; (* V *)
END_VAR
```

MC_BufferMode [▶ 107]

Enable	Allgemeine Software-Freigabe für die Achse.
Enable_Positive	Vorschubfreigabe in positive Richtung. Wirkt nur, wenn *Enable*=TRUE ist.
Enable_Negative	Vorschubfreigabe in negative Richtung. Wirkt nur, wenn *Enable*=TRUE ist.
Override	Geschwindigkeitsoverride in % für alle Fahrbefehle. (0 ≤*Override*≤ 100.0)
BufferMode	Der BufferMode wird ausgewertet, wenn *Enable* zurückgesetzt wird. Der Mode *MC_Aborting* bewirkt ein sofortiges Abschalten der Achsfreigabe. Anderenfalls, z. B. bei Mode *MC_Buffered* wartet der Baustein, bis die Achse keinen Auftrag mehr ausführt.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Status    : BOOL; (* B *)
Busy      : BOOL; (* V *)
Active    : BOOL; (* V *)
Error     : BOOL; (* B *)
ErrorID   : UDINT; (* E *)
END_VAR
```

Status	Status=TRUE zeigt an, dass die Achse betriebsbereit ist.
Busy	Der *Busy*-Ausgang ist TRUE, solange der Funktionsbaustein mit*Enable* TRUE aufgerufen wird
Active	Active zeigt an, dass das Kommando ausgeführt wird.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

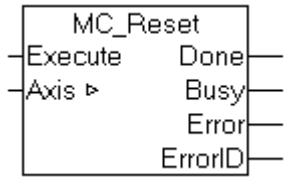
Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.1.2 MC_Reset



Mit dem Funktionsbaustein *MC_Reset* wird ein Reset einer Achse durchgeführt.

MC_Reset führt zunächst einen Reset der NC-Achse durch. In vielen Fällen führt das auch zu einem Reset eines angeschlossenen Antriebsgerätes. Abhängig vom Bussystem oder Antriebstypen kann in einigen Fällen ein separater Reset des Antriebsgerätes notwendig sein.

Eingänge

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn der Reset erfolgreich ausgeführt erreicht wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für ein neues Kommando. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.1.3 MC_SetPosition

MC_SetPosition setzt die aktuelle Achsposition auf einen parametrierbaren Wert.

Im absoluten Modus wird die Istposition auf den parametrierten absoluten Wert *Position* gesetzt. Im relativen Modus wird die Istposition um den parametrierten Wert *Position* verschoben. In beiden Fällen wird die Sollposition der Achse so gesetzt, dass ein eventuell vorhandener Schleppfehler erhalten bleibt. Falls der Schleppfehler gelöscht werden soll, so ist das über den Schalter *Options.ClearPositionLag* möglich.

Der relative Modus ist geeignet, die Achsposition während der Fahrt zu ändern.

Eingänge

```
VAR_INPUT
Execute  : BOOL;
Position : LREAL;
Mode     : BOOL; (* RELATIVE=True, ABSOLUTE=False (Default)*)
Options  : ST_SetPositionOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Position	Positionswert auf den die Achsposition gesetzt werden soll. Im absoluten Modus wird die Istposition auf diesen Wert gesetzt, im relativen Modus wird sie um diesen Wert verschoben.

Mode	Die Achsposition wird auf einen absoluten Wert gesetzt, wenn *Mode*=FALSE ist. Anderenfalls wird die Achsposition relativ um den angegebenen Wert *Position* verändert. Der relative Modus ist geeignet, um die Position einer Achse während der Fahrt anzupassen.	
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.	
Options.	**ClearPositionLag**	ClearPositionLag kann optional gesetzt werden, falls Soll- und Istposition auf den gleichen Wert gesetzt werden sollen. Damit wird der Schleppfehler gelöscht.
Options.	**SelectEncoderIndex**	SelectEncoderIndex kann optional gesetzt werden, falls eine Achse mit mehreren Encodern verwendet wird und die Position eines bestimmten Encoders (*Options.EncoderIndex*) gesetzt werden soll.
Options.	**EncoderIndex**	EncoderIndex gibt den Encoder (0..n) an falls *SelectEncoderIndex* TRUE ist.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE wenn die Position erfolgreich gesetzt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

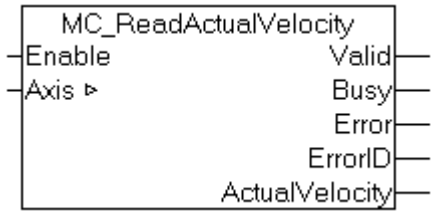
```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2 Status und Parameter

6.2.1 MC_ReadActualVelocity



Mit dem Funktionsbaustein *MC_ReadActualVelocity* kann die aktuelle Position der Achse gelesen werden.

Eingänge

```
VAR_INPUT
Enable   : BOOL;
END_VAR
```

Enable	Solange *Enable* aktiv ist, wird das Kommando ausgeführt.

Ausgänge

```
VAR_OUTPUT
Valid          : BOOL;
Busy           : BOOL;
Error          : BOOL;
ErrorID        : UDINT;
ActualVelocity : LREAL;
END_VAR
```

Valid	Zeigt an, dass der *ActualVelocity* gültig ist.
Busy	Zeigt an, dass der Baustein aktiv ist.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten *Error*-Ausgang die Fehlernummer
ActualVelocity	Augenblickliche Geschwindigkeit der Achse

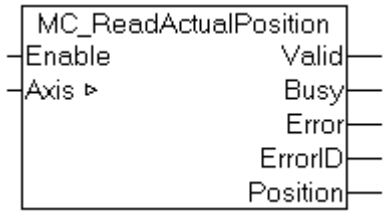
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.2 MC_ReadActualPosition



Mit dem Funktionsbaustein *MC_ReadActualPosition* kann die aktuelle Position der Achse gelesen werden.

Eingänge

```
VAR_INPUT
Enable   : BOOL;
END_VAR
```

Enable	Solange *Enable* aktiv ist, wird das Kommando ausgeführt.

Ausgänge

```
VAR_OUTPUT
Valid    : BOOL;
Busy     : BOOL;
Error    : BOOL;
ErrorID  : UDINT;
Position : LREAL;
END_VAR
```

Valid	Zeigt an, dass der Ausgang *Position* gültig ist.
Busy	Zeigt an, dass der Baustein aktiv ist.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten *Error*-Ausgang die Fehlernummer
Position	Augenblickliche Position der Achse

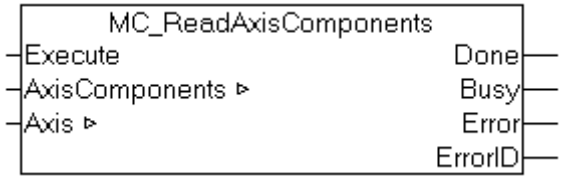
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.3 MC_ReadAxisComponents



Mit dem Funktionsbaustein *MC_ReadAxisComponents* werden Informationen zu den Unterelementen Encoder, Drive und Controller einer Achse gelesen.

Mit "Achse" ist in diesem Fall die TwinCAT NC Achse und nicht der Antrieb gemeint.

Eingänge

```
VAR_INPUT
Execute  : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Wird TRUE, wenn die Parameter erfolgreich gelesen wurden.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

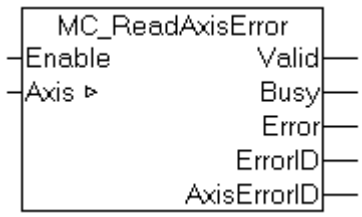
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.4 MC_ReadAxisError



MC_ReadAxisError liest den Achsfehler einer Achse.

Eingänge

```
VAR_INPUT
Enable  : BOOL; (* B *)
END_VAR
```

Enable	Solange *Enable* aktiv ist, wird der Achsfehler am Ausgang *AxisErrorID* ausgegeben

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Valid       : BOOL; (* B *)
Busy        : BOOL; (* E *)
Error       : BOOL; (* B *)
ErrorID     : DWORD; (* B *)
AxisErrorID : DWORD; (* B *)
END_VAR
```

Valid	Der am Ausgang AxisErrorID signalisierte Fehler ist gültig
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Enable* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
AxisErrorID	Fehlernummer der Achse

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

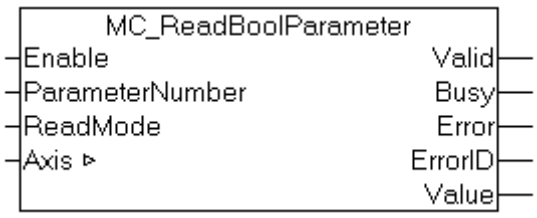
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.5 MC_ReadBoolParameter



Mit dem Funktionsbaustein *MC_ReadBoolParameter* wird ein boolescher Parameter der Achse gelesen.

Mit "Achse" ist in diesem Fall die TwinCAT NC Achse und nicht der Antrieb gemeint.

Eingänge

```
VAR_INPUT
Enable          : BOOL; (* B *)
ParameterNumber : MC_AxisParameter; (* B *)
ReadMode        : E_ReadMode (* V *)
END_VAR
```

E_ReadMode [▶ 112] MC_AxisParameter [▶ 114]

Enable	Solange *Enable* aktiv ist, wird das Kommando ausgeführt.
ParameterNumber	Nummer [▶ 114] des zu lesenden Parameters.
ReadMode	Lesemodus [▶ 112] des zu lesenden Parameters (einmalig oder zyklisch).

Ausgänge

```
VAR_OUTPUT
Valid   : BOOL; (* B *)
Busy    : BOOL; (* E *)
Error   : BOOL; (* B *)
ErrorID : DWORD;(* E *)
Value   : BOOL; (* B *)
END_VAR
```

Valid	Der am Ausgang *Value* signalisierte Wert ist gültig
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Enable* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
Value	Hier wird der gelesene boolesche Wert angezeigt.

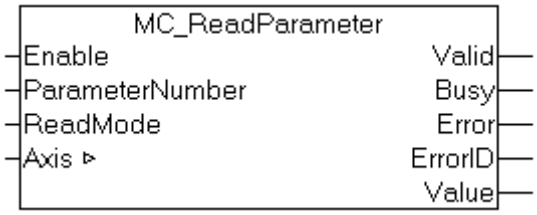
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.6 MC_ReadParameter



Mit dem Funktionsbaustein *MC_ReadParameter* wird ein Parameter der Achse gelesen.

Mit "Achse" ist in diesem Fall die TwinCAT NC Achse und nicht der Antrieb gemeint.

Eingänge

```
VAR_INPUT
Enable          : BOOL; (* B *)
ParameterNumber : MC_AxisParameter; (* B *)
ReadMode        : E_ReadMode (* V *)
END_VAR
```

E_ReadMode [▶ 112] MC_AxisParameter [▶ 114]

Enable	Solange *Enable* aktiv ist, wird das Kommando ausgeführt.
ParameterNumber	Nummer [▶ 114] des zu lesenden Parameters.
ReadMode	Lesemodus [▶ 112] des zu lesenden Parameters (einmalig oder zyklisch).

Ausgänge

```
VAR_OUTPUT
Valid   : BOOL;  (* B *)
Busy    : BOOL;  (* E *)
Error   : BOOL;  (* B *)
ErrorID : DWORD; (* E *)
Value   : LREAL; (* B *)
END_VAR
```

Valid	Der am Ausgang Value signalisierte Wert ist gültig
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Enable* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
Value	Hier wird der gelesene Wert angezeigt.

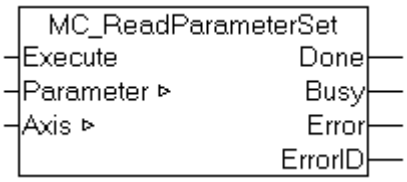
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.7 MC_ReadParameterSet



Mit dem Funktionsbaustein *MC_ReadParameterSet* kann der gesamte Parameter-Satz einer Achse gelesen werden.

Mit "Achse" ist in diesem Fall die TwinCAT NC Achse und nicht der Antrieb gemeint.

Eingänge

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Wird TRUE, wenn die Parameter erfolgreich gelesen wurden.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

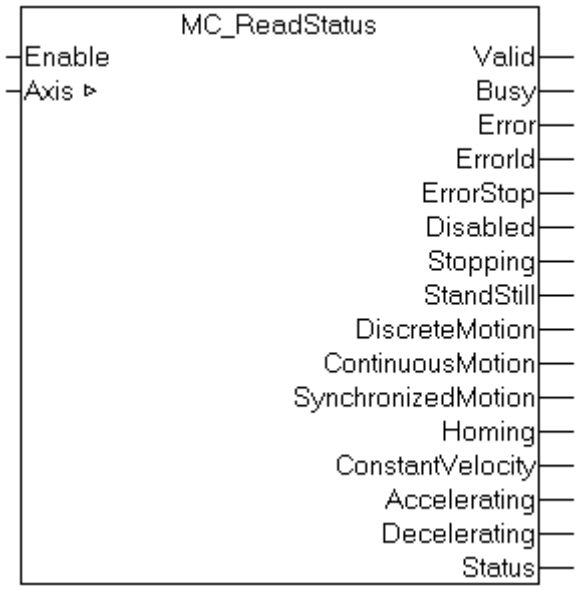
Ein/Ausgänge

```
VAR_IN_OUT
Parameter : ST_AxisParameterSet;
Axis      : AXIS_REF;
END_VAR
```

Parameter	Parameter-Datenstruktur [▶ 116] in die die Parameter eingelesen werden
Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.8 MC_ReadStatus



MC_ReadStatus ermittelt den aktuellen Betriebszustand einer Achse und signalisiert ihn an den Ausgängen des Bausteins.

Der aktualisierte Betriebszustand wird zusätzlich in der Ausgangsdatenstruktur *Status* und in der Achsdatenstruktur *Axis.Status* abgelegt. Dadurch ist es ausreichend, den Betriebszustand einmalig am Anfang jedes SPS-Zyklus zu lesen und im weiteren Programmverlauf auf *Axis.Status* zuzugreifen.

Die *Axis*-Variable (Typ AXIS_REF) enthält bereits eine Instanz des Funktionsbausteins *MC_ReadStatus*. Dadurch ist es einfach möglich, den Betriebszustand einer Achse am Anfang eines SPS-Zyklus durch den Aufruf von Axis.ReadStatus zu aktualisieren.

Beispiel:

```
PROGRAM MAIN
VAR
Axis1   : AXIS_REF
END_VAR

(* call the read status function *)
Axis1.ReadStatus;
```

Eingänge

```
VAR_INPUT
Enable  : BOOL;
END_VAR
```

Enable	Solange *Enable*=TRUE ist, wird der Betriebszustand der Achse mit jedem Aufruf des Bausteins aktualisiert.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Valid               : BOOL;
Busy                : BOOL;
Error               : BOOL;
ErrorId             : UDINT;
(* motion control statemachine states: *)
ErrorStop           : BOOL;
Disabled            : BOOL;
Stopping            : BOOL;
StandStill          : BOOL;
DiscreteMotion      : BOOL;
ContinuousMotion    : BOOL;
SynchronizedMotion  : BOOL;
Homing              : BOOL;
(* additional status *)
ConstantVelocity    : BOOL;
Accelerating        : BOOL;
Decelerating        : BOOL;
(* status data structure *)
Status              : ST_AxisStatus;
END_VAR
```

Valid	Zeigt an, dass der an den weiteren Ausgängen angezeigte Betriebszustand der Achse gültig ist.
Busy	Zeigt an, dass der Baustein aktiv ist.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
ErrorStop	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
Disabled	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
Stopping	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
StandStill	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
DiscreteMotion	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
ContinousMotion	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
SynchronizedMotion	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
Homing	Zustand der Achse gemäß des PlcOpen Zustandsdiagramms [▶ 12]
ConstantVelocity	Die Achse fährt mit konstanter Geschwindigkeit
Acceleration	Die Achse beschleunigt.
Decelerating	Die Achse bremst ab.
Status	Erweiterte Status-Datenstruktur [▶ 113] mit weiteren Status-Informationen.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

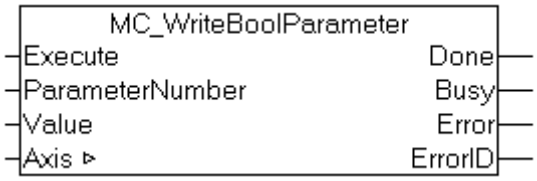
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.9 MC_WriteBoolParameter



Mit dem Funktionsbaustein *MC_WriteBoolParameter* können boolsche Parameter für die Achse geschrieben werden.

Mit "Achse" ist in diesem Fall die TwinCAT NC Achse und nicht der Antrieb gemeint.

Eingänge

```
VAR_INPUT
Execute         : BOOL;
ParameterNumber : INT;
Value           : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt.
ParameterNumber	Nummer [▶ 114] des zu schreibenden Parameters.
Value	Dieser BOOL Wert wird geschrieben.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Wird TRUE, wenn die Parameter erfolgreich geschrieben wurden.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

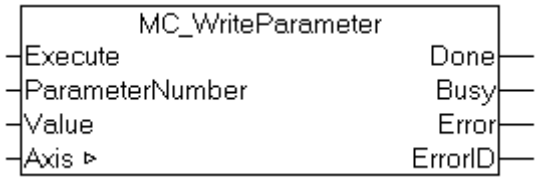
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.2.10 MC_WriteParameter



Mit dem Funktionsbaustein *MC_WriteParameter* können Parameter für die Achse geschrieben werden.

Mit "Achse" ist in diesem Fall die TwinCAT NC Achse und nicht der Antrieb gemeint.

Eingänge

```
VAR_INPUT
Execute         : BOOL;
ParameterNumber : INT;
Value           : LREAL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt.
ParameterNumber	Nummer [▶ 114] des zu schreibenden Parameters.
Value	Dieser LREAL Wert wird geschrieben.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Wird TRUE, wenn die Parameter erfolgreich geschrieben wurden.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.

ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer

Ein/Ausgänge

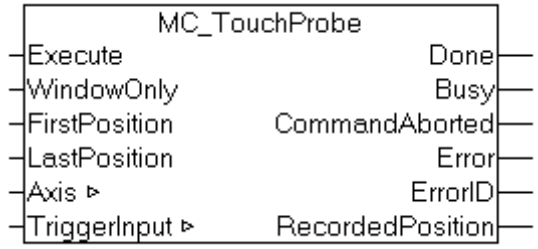
```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.3 Touch Probe

6.3.1 MC_TouchProbe



Der Baustein *MC_TouchProbe* erfasst eine Achsposition zum Zeitpunkt eines digitalen Signals (Messtasterfunktion). Die Position wird üblicherweise nicht direkt in der SPS-Umgebung, sondern über ein externes Hardware-Latch erfasst und ist damit hochgenau und Zykluszeit-unabhängig. Der Baustein steuert diesen Mechanismus und ermittelt die extern erfasste Position.

Voraussetzungen

Voraussetzung für die Positionserfassung ist eine geeignete Encoder-Hardware, die in der Lage ist, die erfasste Position zu latchen. Unterstützt werden beispielsweise SERCOS-Antriebe, Beckhoff AX2000 mit SERCOS- und Lightbus-Schnittstelle und Beckhoff Encoder-Klemmen KL5101. Das digitale Trigger-Signal wird mit dieser Hardware verdrahtet und führt unabhängig vom SPS-Zyklus zum Aufzeichnen der aktuellen Achsposition.

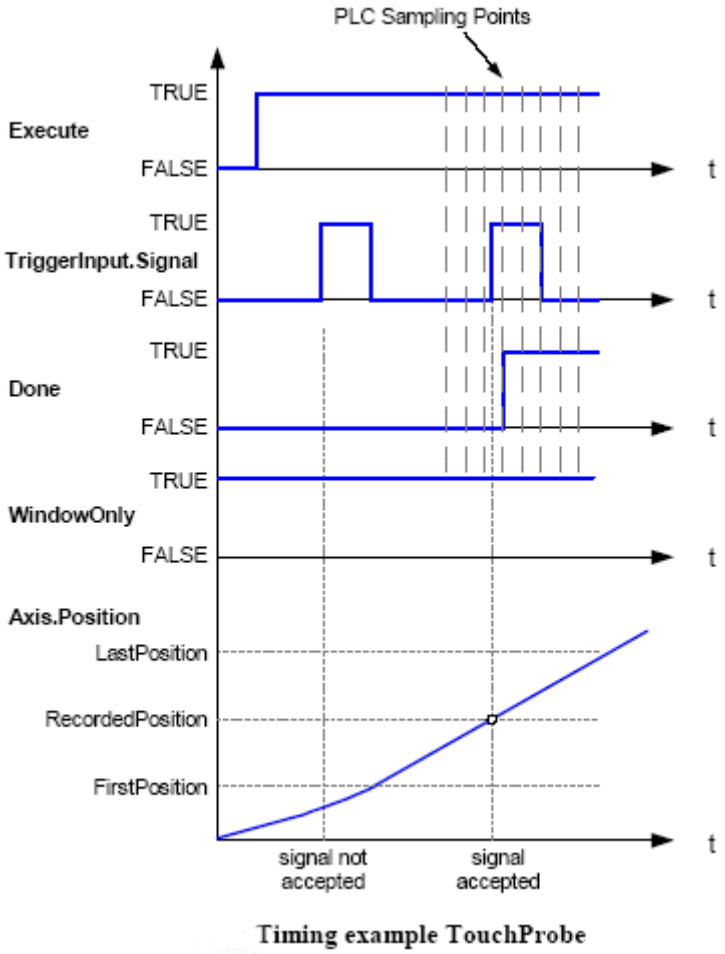
Teilweise müssen diese Endgeräte konfiguriert werden, damit eine Positionserfassung möglich ist. Lesen Sie dazu Messtasterauswertung mit AX2xxx-B200 (Lightbus), Messtasterauswertung mit AX2xxx-B750 (SERCOS), AX5000 Probe Unit und AX5000 Funktion einer Probe Unit.

Einschränkungen

MC_TouchProbe kann unabhängig von der verwendeten Hardware nur eine Flanke einer Probe Unit gleichzeitig erfassen. Sollen beispielsweise beide Flanken erfasst werden, so muss der Baustein nach der ersten Flanke mit geänderter Parametrierung erneut gestartet werden. Kurz aufeinanderfolgende Flanken können somit nicht erfasst werden. Um dieses Problem zu umgehen, wird auf den erweiterten Baustein MC_TouchProbe_V2 [▶ 39] verwiesen.

Nachdem ein Messtasterzyklus durch eine steigende Flanke am *Execute* Eingang gestartet wurde, wird dieser erst beendet, wenn die Ausgänge *Done*, *Error* oder *CommandAborted* TRUE werden. Soll der Vorgang zwischenzeitlich abgebrochen werden, so muss der Baustein MC_AbortTrigger [▶ 42] mit derselben TriggerInput [▶ 118] Datenstruktur aufgerufen werden. Anderenfalls kann kein neuer Zyklus gestartet werden.

Signalverlauf



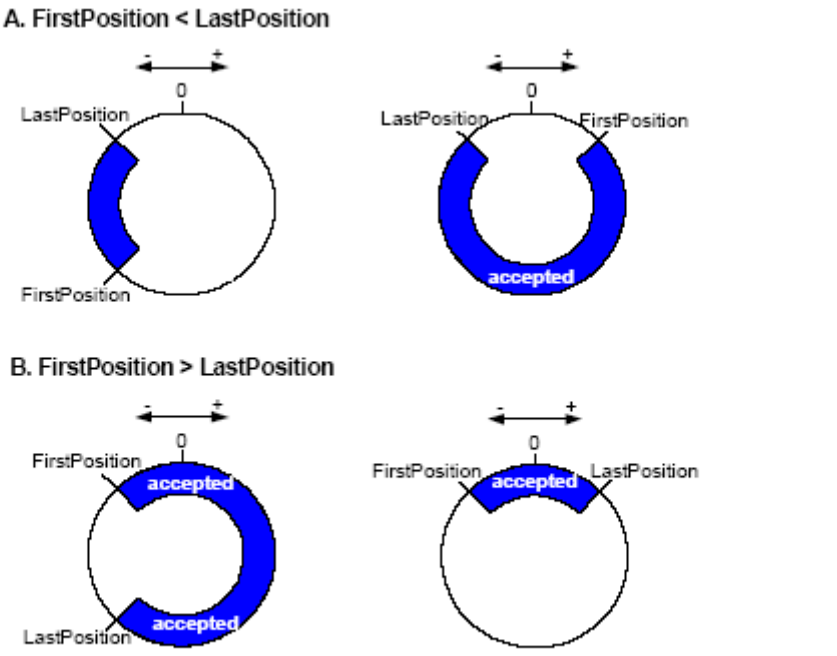
Timing example TouchProbe

Eingänge

```
VAR_INPUT
Execute       : BOOL;
WindowOnly    : BOOL;
FirstPosition : LREAL;
LastPosition  : LREAL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt und das externe Positionslatch wird aktiviert.
WindowOnly	Wenn diese Option aktiv ist, wird nur eine Position innerhalb des Fensters zwischen *FirstPosition* und LastPosition erfasst. Positionen außerhalb des Fensters werden verworfen und das externe Positionslatch wird automatisch neu aktiviert. Erst wenn die erfasste Position innerhalb des Fensters liegt, wird *Done* TRUE. Das Erfassungsfenster kann absolut oder modulo interpretiert werden. Dazu ist das Flag ModuloPositions [▶ 118] in der Struktur TriggerInput [▶ 118] entsprechend zu setzen. Bei absoluten Positionen gibt es exakt ein Fenster. Bei Modulo-Positionen wiederholt sich das Fenster innerhalb des in den Achsparametern festgelegten Modulozyklus (z. B. 0 bis 360 Grad).

FirstPosition	Anfangsposition des Erfassungsfensters, wenn WindowOnly TRUE ist. Diese Position kann absolut oder modulo interpretiert werden. Dazu ist in der Struktur *TriggerInput* (siehe unten) das Flag ModuloPositions [▶ 118] entsprechend zu setzen.
LastPosition	Endposition des Erfassungsfensters, wenn WindowOnly TRUE ist. Diese Position kann absolut oder modulo interpretiert werden. Dazu ist in der Struktur *TriggerInput* (siehe unten) das Flag ModuloPositions [▶ 118] entsprechend zu setzen.



examples of windows, where trigger events are accepted (for modulo axes)

Ausgänge

```
VAR_OUTPUT
Done              : BOOL;
Busy              : BOOL;
CommandAborted    : BOOL;
Error             : BOOL;
ErrorID           : UDINT;
RecordedPosition  : LREAL;
END_VAR
```

Done	Wird TRUE, wenn eine Achsposition erfolgreich erfasst wurde. Die Position wird am Ausgang *RecordedPosition* ausgegeben.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
CommandAborted	Wird TRUE wenn der Vorgang von außen, z. B. durch den Aufruf von MC_AbortTrigger [▶ 42], abgebrochen wurde.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
RecordedPosition	Erfasste Achsposition zum Zeitpunkt des Trigger-Signals

Ein/Ausgänge

```
VAR_IN_OUT
Axis         : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Achsdatenstruktur [▶ 105]
TriggerInput	Datenstruktur TRIGGER. REF [▶ 118] zur Beschreibung der Trigger-Quelle

6.3.2 MC_TouchProbe_V2

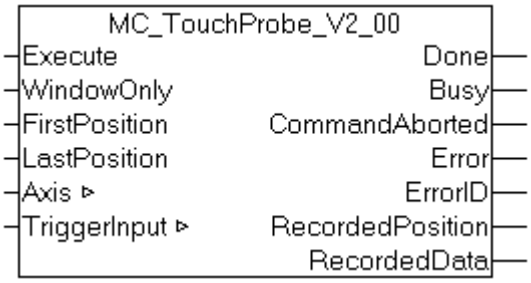


Abb. 1: MC_TouchProbe_V2_00

Der Baustein *MC_TouchProbe_V2* erfasst eine Achsposition zum Zeitpunkt eines digitalen Signals (Messtasterfunktion). Die Position wird üblicherweise nicht direkt in der SPS-Umgebung, sondern über ein externes Hardware-Latch erfasst und ist damit hochgenau und Zykluszeit-unabhängig. Der Baustein steuert diesen Mechanismus und ermittelt die extern erfasste Position.

Die Funktion des Bausteins *MC_TouchProbe_V2* ist ähnlich der des Bausteins *MC_TouchProbe*. Mit mehreren Instanzen ist es aber möglich, bis zu zwei Probe-Units eines Antriebs gleichzeitig zu bedienen und es kann parallel die steigende und fallende Signalflanke mit jeweils einer Instanz erfasst werden. Weiterhin steht ein Continuous-Mode zur Verfügung der ohne erneute Aktivierung aufeinanderfolgende Signalflanken auswertet.

Voraussetzungen

- TwinCAT ab Version 2.11 R2 Build 2022 (davor MC_TouchProbe [▶ 36] verwenden)

Voraussetzung für die Positionserfassung ist eine geeignete Encoder-Hardware, die in der Lage ist, die erfasste Position zu latchen. Unterstützt werden:

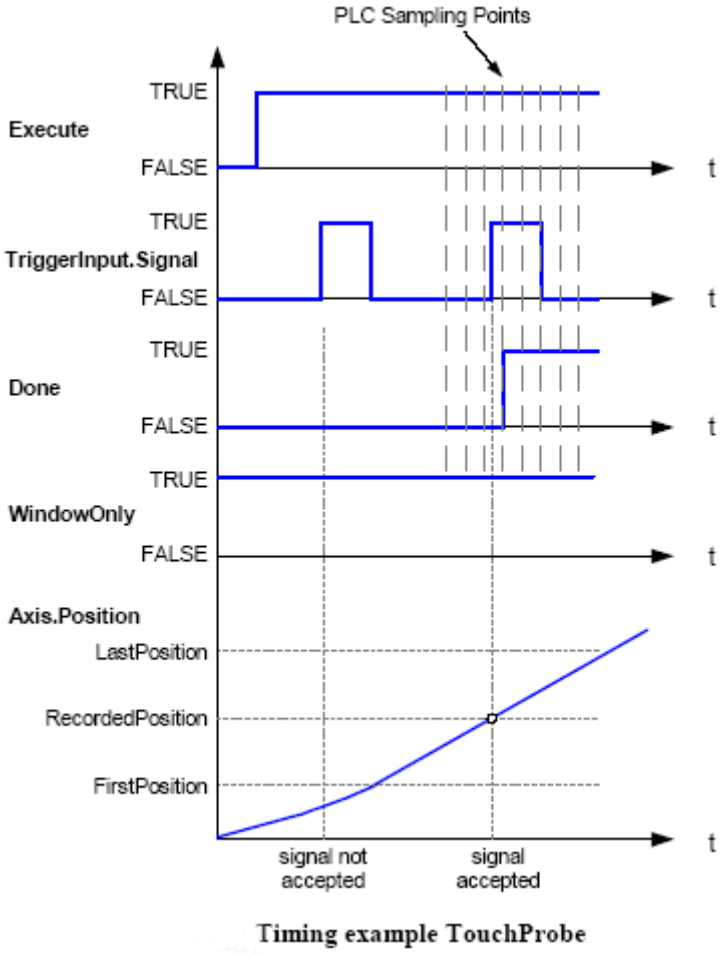
- SERCOS Antriebe
  Der Antrieb muss abweichend von MC_TouchProbe mit einer erweiterten Schnittstelle konfiguriert werden, bei der die Parameter S-0-0405 bzw. S-0-0406 im Prozessabbild enthalten sind. Siehe dazu ...
- EtherCAT SoE Antriebe (z. B. AX5000)
  Der Antrieb muss abweichend von MC_TouchProbe mit einer erweiterten Schnittstelle konfiguriert werden, bei der die Parameter S-0-0405 bzw. S-0-0406 im Prozessabbild enthalten sind. Siehe dazu ...
- EtherCAT CoE Antriebe
  Der Antrieb muss mit dem Parameter 0x60B9 (Touch Probe Status) im Prozessabbild konfiguriert werden.
- EL5101, KL5101
  Latchen der C-Spur und des digitalen Eingangs ist möglich. Mit dieser Hardware kann zeitgleich nur ein Signal bzw. eine Flanke erfasst werden. Der Continuous-Mode wird nicht unterstützt.

Das digitale Trigger-Signal wird mit dieser Hardware verdrahtet und führt unabhängig vom SPS-Zyklus zum Aufzeichnen der aktuellen Achsposition.

Teilweise müssen diese Endgeräte konfiguriert werden, damit eine Positionserfassung möglich ist. Beckhoff EtherCAT-Antriebe können mit dem SystemManager konfiguriert werden. Hierbei ist zu beachten, dass die Probe-Unit mit dem erweiterten Interface "Extended NC Probe Unit" konfiguriert werden muss.

Nachdem ein Messtasterzyklus durch eine steigende Flanke am *Execute* Eingang gestartet wurde, wird dieser erst beendet, wenn die Ausgänge *Done*, *Error* oder *CommandAborted* TRUE werden. Soll der Vorgang zwischenzeitlich abgebrochen werden, so muss der Baustein MC_AbortTrigger_V2 [▶ 43] mit derselben TriggerInput [▶ 118] Datenstruktur aufgerufen werden. Anderenfalls kann kein neuer Zyklus gestartet werden.

Signalverlauf



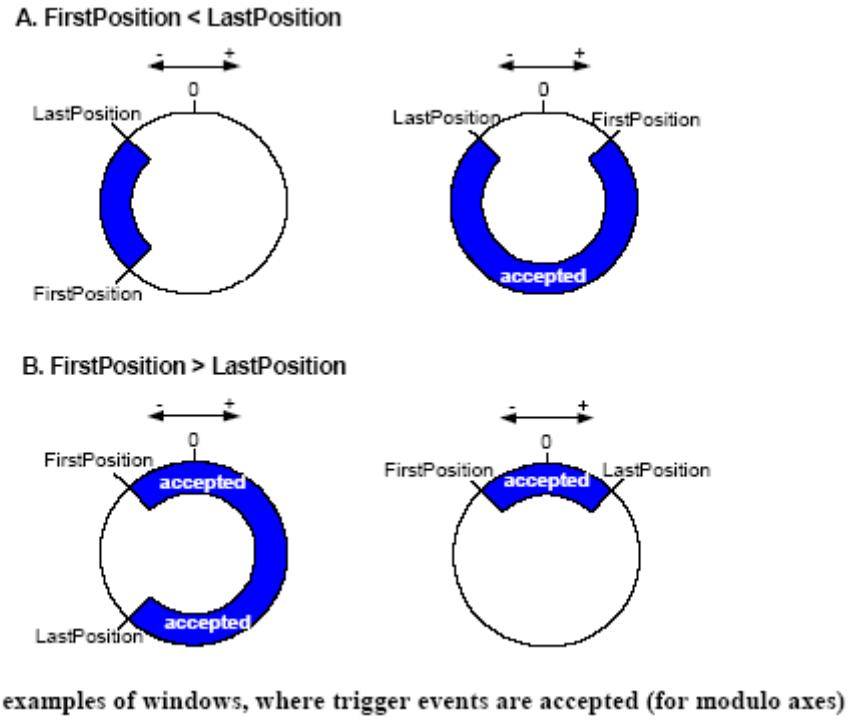
Timing example TouchProbe

Eingänge

```
VAR_INPUT
Execute : BOOL;
WindowOnly : BOOL;
FirstPosition : LREAL;
LastPosition  : LREAL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt und das externe Positionslatch wird aktiviert.
WindowOnly	Wenn diese Option aktiv ist, wird nur eine Position innerhalb des Fensters zwischen *FirstPosition* und LastPosition erfasst. Positionen außerhalb des Fensters werden verworfen und das externe Positionslatch wird automatisch neu aktiviert. Erst wenn die erfasste Position innerhalb des Fensters liegt, wird *Done* TRUE. Das Erfassungsfenster kann absolut oder modulo interpretiert werden. Dazu ist das Flag ModuloPositions [▶ 118] in der Struktur TriggerInput

	[▶ 118] entsprechend zu setzen. Bei absoluten Positionen gibt es exakt ein Fenster. Bei Modulo-Positionen wiederholt sich das Fenster innerhalb des in den Achsparametern festgelegten Modulozyklus (z. B. 0 bis 360 Grad).
FirstPosition	Anfangsposition des Erfassungsfensters, wenn WindowOnly TRUE ist. Diese Position kann absolut oder modulo interpretiert werden. Dazu ist in der Struktur *TriggerInput* (siehe unten) das Flag ModuloPositions [▶ 118] entsprechend zu setzen.
LastPosition	Endposition des Erfassungsfensters, wenn WindowOnly TRUE ist. Diese Position kann absolut oder modulo interpretiert werden. Dazu ist in der Struktur *TriggerInput* (siehe unten) das Flag ModuloPositions [▶ 118] entsprechend zu setzen.



examples of windows, where trigger events are accepted (for modulo axes)

Ausgänge

```
VAR_OUTPUT
Done             : BOOL;
Busy             : BOOL;
CommandAborted   : BOOL;
Error            : BOOL;
ErrorID          : UDINT;
RecordedPosition : LREAL;
RecordedData : MC_TouchProbeRecordedData;
END_VAR
```

Done	Wird TRUE, wenn eine Achsposition erfolgreich erfasst wurde. Die Position wird am Ausgang *RecordedPosition* ausgegeben.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
CommandAborted	Wird TRUE wenn der Vorgang von außen, z. B. durch den Aufruf von MC_AbortTrigger [▶ 42], abgebrochen wurde.

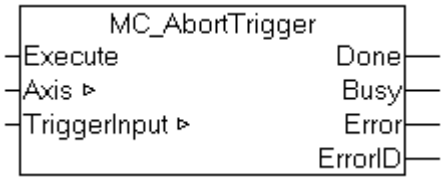
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.
RecordedPosition	Erfasste Achsposition zum Zeitpunkt des Trigger-Signals
RecordedData	Datenstruktur mit ergänzenden Informationen zur erfassten Achsposition zum Zeitpunkt des Trigger-Signals

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Achsdatenstruktur [▶ 105]
TriggerInput	Datenstruktur TRIGGER_REF [▶ 118] zur Beschreibung der Trigger-Quelle

6.3.3 MC_AbortTrigger



Der Baustein *MC_AbortTrigger* bricht einen durch *MC_TouchProbe* gestarteten Messtasterzyklus ab. *MC_TouchProbe* startet einen Messtasterzyklus, indem ein Positionslatch in einer externen Encoder- oder Antriebs-Hardware aktiviert wird. Soll der Vorgang beendet werden, bevor das Trigger-Signal das Positionslatch aktiviert hat, so kann dazu *MC_AbortTrigger* verwendet werden. Wurde der Messtasterzyklus erfolgreich beendet, so ist es nicht notwendig diesen Baustein aufzurufen.

Eingänge

```
VAR_INPUT
Execute  : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt und das externe Positionslatch wird deaktiviert.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

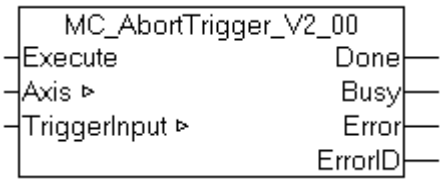
Done	Wird TRUE, sobald der Messtasterzyklus erfolgreich abgebrochen wurde.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

```
VAR_IN_OUT
Axis         : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Achsdatenstruktur [▶ 105]
TriggerInput	Datenstruktur TRIGGER_REF [▶ 118] zur Beschreibung der Trigger-Quelle. Diese Datenstruktur muss vor dem ersten Aufruf des Funktionsbausteins parametriert werden.

6.3.4 MC_AbortTrigger_V2



Der Baustein *MC_AbortTrigger_V2* bricht einen durch *MC_TouchProbe_V2* gestarteten Messtasterzyklus ab. *MC_TouchProbe_V2* startet einen Messtasterzyklus, indem ein Positionslatch in einer externen Encoder- oder Antriebs-Hardware aktiviert wird. Soll der Vorgang beendet werden, bevor das Trigger-Signal das Positionslatch aktiviert hat, so kann dazu *MC_AbortTrigger_V2* verwendet werden. Wurde der Messtasterzyklus erfolgreich beendet, so ist es nicht notwendig diesen Baustein aufzurufen.

Eingänge

```
VAR_INPUT
Execute  : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt und das externe Positionslatch wird deaktiviert.

Ausgänge

```
VAR_OUTPUT
Done     : BOOL;
Busy     : BOOL;
Error    : BOOL;
ErrorID  : UDINT;
END_VAR
```

Done	Wird TRUE, sobald der Messtasterzyklus erfolgreich abgebrochen wurde.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

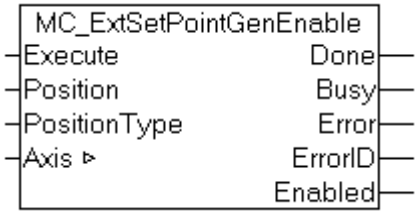
```
VAR_IN_OUT
Axis         : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Achsdatenstruktur [▶ 105]

TriggerInput	Datenstruktur TRIGGER_REF [▶ 118] zur Beschreibung der Trigger-Quelle. Diese Datenstruktur muss vor dem ersten Aufruf des Funktionsbausteins parametriert werden.

6.4 Externer Sollwertgenerator

6.4.1 MC_ExtSetPointGenEnable



Mit dem Funktionsbaustein *MC_ExtSetPointGenEnable* kann der externe Sollwertgenerator einer Achse eingeschaltet werden. Anschließend übernimmt die Achse die Sollwertvorgaben aus ihrem zyklischen Achsinterface [▶ 106] (*Axis.PlcToNc.ExtSetPos, ExtSetVelo, ExtSetAcc* und *ExtSetDirection*).

Ein so genannter *externer Sollwertgenerator* ist üblicherweise ein SPS-Baustein, der zyklische Sollwerte für eine Achse berechnet und somit den in einer NC-Achse enthaltenen *internen Sollwertgenerator* ersetzen kann.

Siehe auch: MC_ExtSetPointGenDisable [▶ 45] und MC_ExtSetPointGenFeed [▶ 46]

Eingänge

```
VAR_INPUT
Execute      : BOOL;
Position     : LREAL;
PositionType : E_PositionType;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt.
Position	Position für Zielpositionsüberwachung. Das Setzten dieser Position bedeutet nicht, dass die Achse zu diese Position verfährt, dafür ist ausschließlich der externe Sollwertgenerator verantwortlich. Vielmehr wird durch das Setzen dieser Position die Zielpositionsüberwachung aktiviert und das Flag Datentyp ST_AxisStatus [▶ 113] wird TRUE sobald diese Position erreicht wird.
PositionType	Positionstyp [▶ 120] - *POSITIONTYPE_ABSOLUTE* oder *POSITIONTYPE_RELATIVE*

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
Enabled : BOOL;
END_VAR
```

Done	Wird TRUE, wenn der Befehl erfolgreich abgesetzt wurde.

Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
Enabled	Enabled zeigt unabhängig von der Funktionsausführung den aktuellen Zustand des externen Sollwertgenerators an.

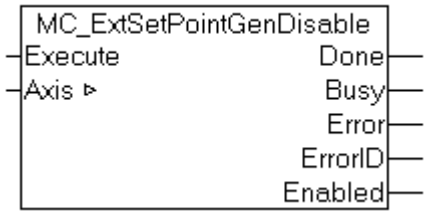
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur [▶ 105]

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.4.2 MC_ExtSetPointGenDisable



Mit dem Funktionsbaustein *MC_ExtSetPointGenDisable* kann der externe Sollwertgenerator einer Achse ausgeschaltet werden. Anschließend übernimmt die Achse nicht mehr die Sollwertvorgaben aus ihrem zyklischen Achsinterface [▶ 106] (*Axis.PlcToNc.ExtSetPos, ExtSetVelo, ExtSetAcc* und *ExtSetDirection*)

Ein so genannter *externer Sollwertgenerator* ist üblicherweise ein SPS-Baustein, der zyklische Sollwerte für eine Achse berechnet und somit den in einer NC-Achse enthaltenen *internen Sollwertgenerator* ersetzen kann.

Siehe auch: MC_ExtSetPointGenEnable [▶ 44] und MC_ExtSetPointGenFeed [▶ 46]

Eingänge

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt.

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
Enabled : BOOL;
END_VAR
```

Done	Wird TRUE, wenn der Befehl erfolgreich ausgeführt wurde.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer
Enabled	Enabled zeigt unabhängig von der Funktionsausführung den aktuellen Zustand des externen Sollwertgenerators an.

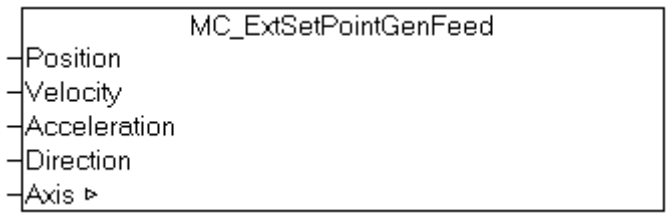
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.4.3 MC_ExtSetPointGenFeed



Mit der Funktion *MC_ExtSetPointGenFeed* werden die Sollwerte eines externen Sollwertgenerators in eine Achse eingespeist. Die Funktion kopiert die Daten instantan in das zyklische Achsinterface [▶ 106] (*fExtSetPos, fExtSetVelo, fExtSetAcc* und *nExtSetDirection*) der Achse. Das Funktionsergebnis *MC_ExtSetPointGenFeed* ist ungenutzt und daher immer FALSE.

Ein so genannter *externer Sollwertgenerator* ist üblicherweise ein SPS-Baustein, der zyklische Sollwerte für eine Achse berechnet und somit den in einer NC-Achse enthaltenen *internen Sollwertgenerator* ersetzen kann.

Siehe auch: MC_ExtSetPointGenEnable [▶ 44] und MC_ExtSetPointGenDisable [▶ 45]

Eingänge

```
VAR_INPUT
Position     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Direction    : DINT;
END_VAR
```

Position	Sollposition aus einem externen Sollwertgenerator
Velocity	Sollgeschwindigkeit aus einem externen Sollwertgenerator
Acceleration	Sollbeschleunigung aus einem externen Sollwertgenerator

Direction	Sollrichtung aus einem externen Sollwertgenerator. (-1 = negative Richtung, 0 = Stillstand, 1 = positive Richtung)

Ein/Ausgänge

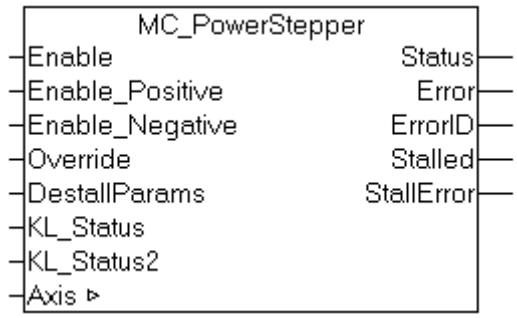
```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.5 Spezielle Erweiterungen

6.5.1 MC_PowerStepper



Mit dem Funktionsbaustein *MC_PowerStepper* werden die Freigaben für eine Achse gesetzt. Dazu wird intern ein *MC_Power*-Baustein verwendet. Zusätzlich erkennt der *MC_PowerStepper* die bei Schrittmotoren im Überlastfall auftretenden Stall-Situationen und bietet geeignete Gegenmaßnahmen an. Die Statusbits einer KL2531 oder KL2541 Klemme werden überwacht und die dort signalisierten Fehler an die NC gemeldet.

Weitere Erläuterungen im Anhang [▶ 48].

Eingänge

```
VAR_INPUT
Enable          : BOOL;
Enable_Positive : BOOL;
Enable_Negative : BOOL;
Override        : LREAL;
DestallParams   : ST_PowerStepperStruct;
KL_Status       : USINT;
KL_Status2      : UINT;
END_VAR
```

Enable	NC-Reglerfreigabe für die Achse.
Enable_Positive	NC-Vorschubfreigabe in positiver Richtung.
Enable_Negative	NC-Vorschubfreigabe in negativer Richtung.
Override	Overridewert in Prozent (z. B. als 68.123%)
DestallParams	Hier [▶ 115] werden die Funktionen des Bausteins freigegeben und ihre Arbeitsregeln festgelegt. ST_PowerStepperStruct [▶ 115]

KL_Status	Das Statusbyte einer Klemme des Typs KL2531 oder KL2541.
KL_Status2	Das Statuswort einer Klemme des Typs KL2531 oder KL2541.

Ausgänge

```
VAR_OUTPUT
Status     : BOOL;
Error      : BOOL;
ErrorID    : UDINT;
Stalled    : BOOL;
StallError : BOOL;
END_VAR
```

Status	Wird TRUE, wenn die Freigaben erfolgreich gesetzt wurden.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.
Stalled	no description
StallError	no description

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

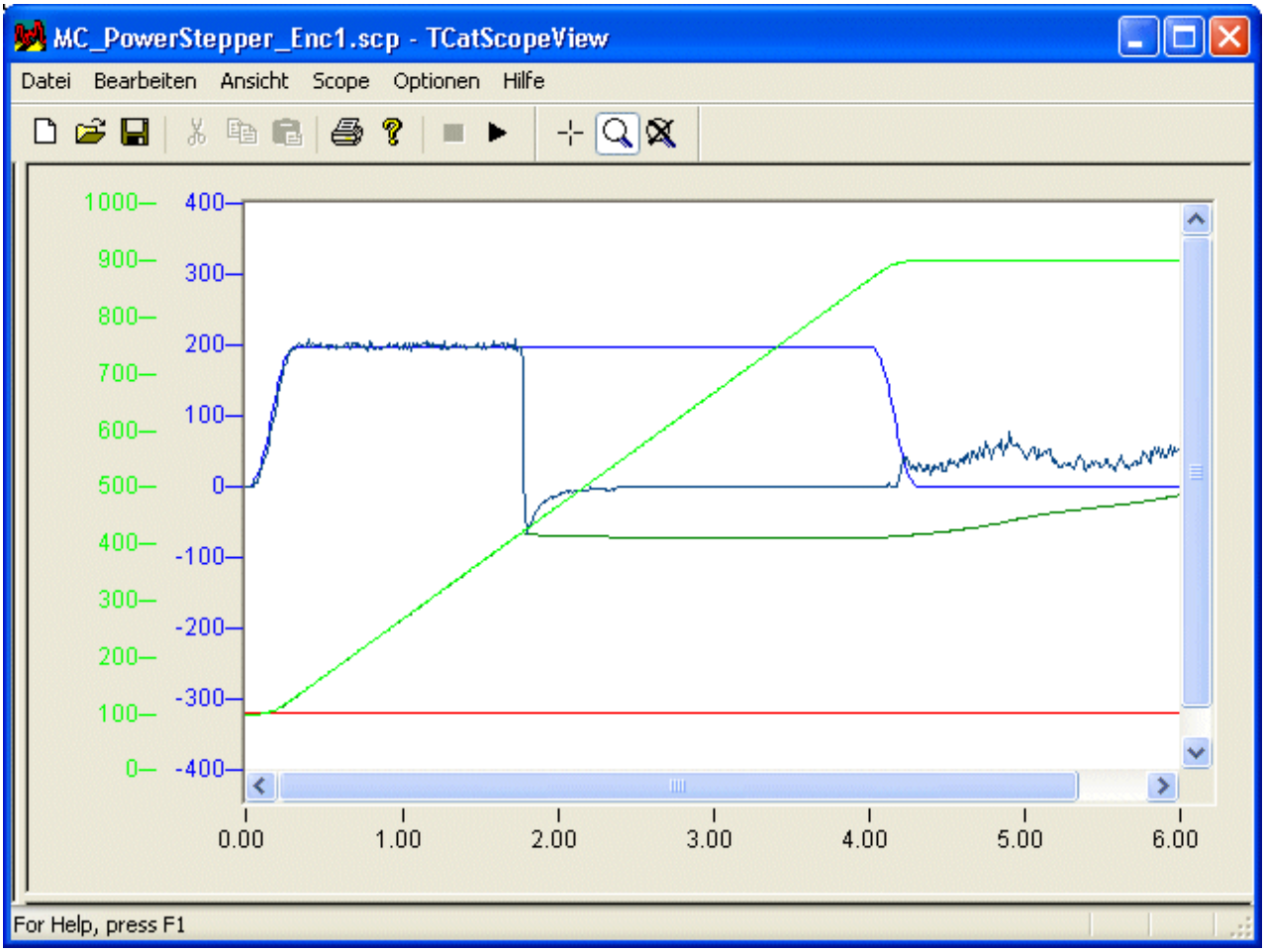
6.5.2 Hinweise zu MC_PowerStepper

Mit dem Funktionsbaustein MC_PowerStepper [▶ 47] werden die Freigaben und der Override für eine Achse gesetzt. Dazu wird intern ein MC_Power [▶ 21] Baustein verwendet. Zusätzlich erkennt der *MC_PowerStepper* die bei Schrittmotoren im Überlastfall auftretenden Stall-Situationen und bietet geeignete Gegenmaßnahmen an. Die Stausbits einer KL2531 oder KL2541 Klemme werden überwacht und die dort signalisierten Fehler an die NC gemeldet.

Schrittmotor und Synchron-Servo: Gemeinsamkeiten und Unterschiede

Beide Motortypen verwenden ein elektromagnetisches und ein permanentmagnetisches Feld, um durch deren Zusammenspiel eine Antriebskraft zu erzeugen. Während jedoch beim Servo eine kostenaufwändige Sensorik eingesetzt wird, um die Ausrichtung der Felder gezielt zu kontrollieren (rotorlageorientierte Bestromung) wird beim Schrittmotor auf diese Orientierung verzichtet. Dadurch ist eine deutliche Kosteneinsparung erzielbar. Allerdings entsteht so die Möglichkeit, dass der Motor durch eine äußere Kraft über den Punkt des maximalen von ihm erzeugten Moments hinaus bewegt wird. Da das elektromagnetisch erzeugte Feld dies nicht berücksichtigt wird er bei weiter steigender Auslenkung jetzt ein absinkendes Rückstellmoment erzeugen. Dies führt dazu, dass ab einer Auslenkung von mehr als einem halben Polspiel das Rückstellmoment im Vorzeichen wechselt und den Motor zum nächsten Pol mit entsprechender Richtung zieht. Je nach jetzt geltenden Bedingungen kann der Motor jetzt in der neuen Orientierung einrasten (der so genannte Schrittverlust ist eingetreten) oder auch hier den gleichen Vorgang wiederholen. Letzteres wird als Stalling bezeichnet und wird vor allem dann auftreten, wenn der Motor mit typischen Frequenzen des aktiven Fahrbetriebs bestromt wird.

Beispiel 1: Ein mit einem Encoder ausgerüsteter Schrittmotor wird mit für Servos üblichen Parametern mit der NC PTP verfahren.

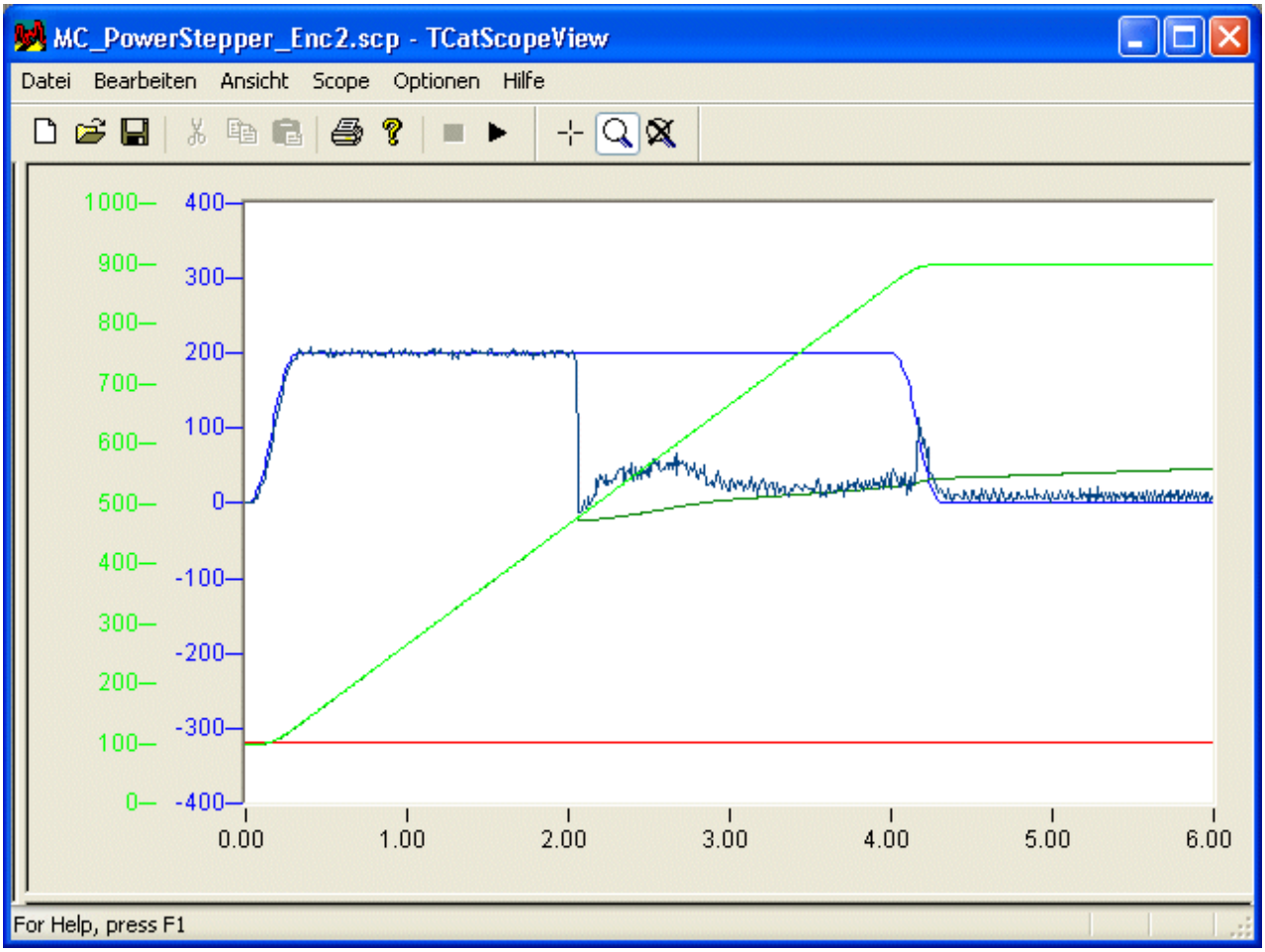


bStalled
SetVelo
ActualPos
ActualVelo
SetPos

Bei etwa 1.8 Sekunden wird die Achse von einem Hindernis kurzzeitig blockiert. Obwohl die Achse anschließend frei beweglich ist kann sie dem Geschwindigkeitssollwert nicht folgen, sondern steht mit erheblicher Geräuschentwicklung und ohne ein erkennbares Moment zu erzeugen still. Erst nachdem der Profilgenerator sein Fahrziel erreicht hat sinkt die Summe aus Soll- und Korrekturgeschwindigkeit ab. Der Motor setzt sich in diesem Beispiel in unregelmäßiger Weise in Bewegung. Schon ein geringes Lastmoment würde dies jedoch verhindern. Die einzige Lösung würde hier ein MC_Reset [▶ 22] und eine angemessene Beruhigungszeit sein. Anschließend müsste die Achse durch die Applikation erneut gestartet werden. Dabei würden im Achsinterface diverse Zustandsbits reagieren. Dies ist in der Applikation entsprechend zu berücksichtigen, da es sonst zu fehlerhaften Reaktionen im Maschinenablauf kommen kann.

Erste Maßnahme: Reglerbegrenzung

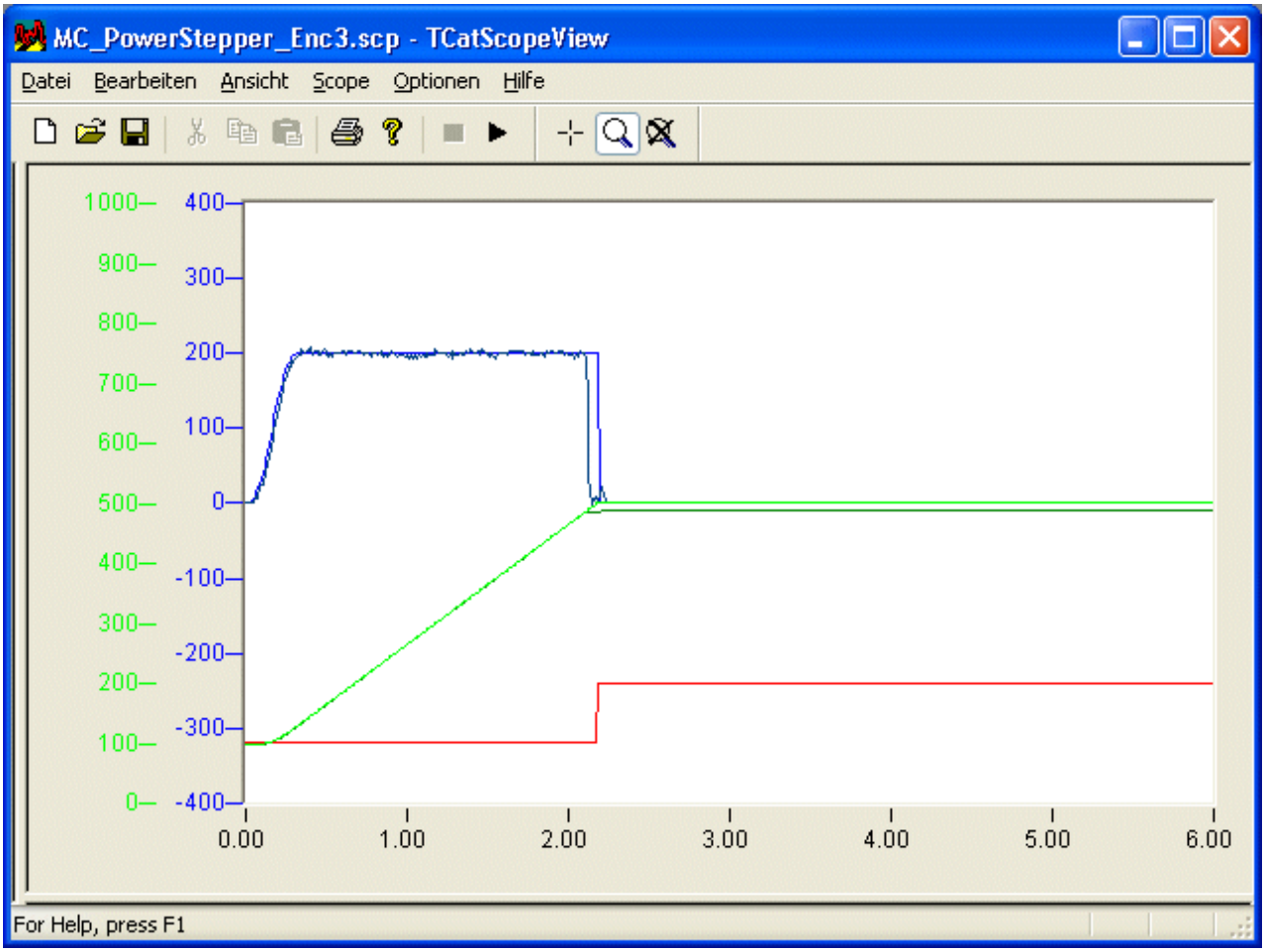
Wenn in der oben beschriebenen Situation die Ausgabe des Lagereglers auf einen ausreichend kleinen Wert wie z.B. 2% begrenzt wird ergibt sich das nachfolgende Bild.



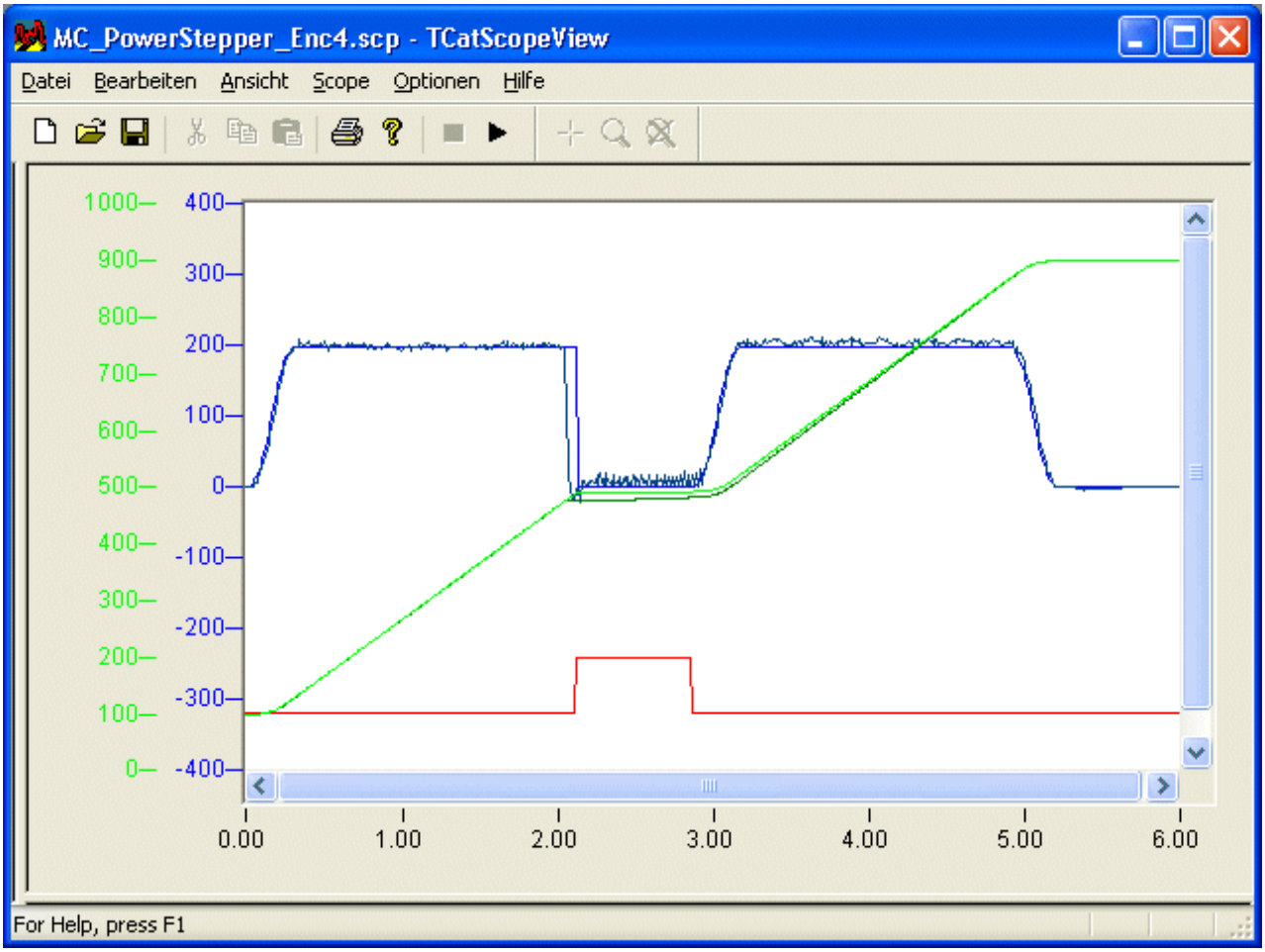
Auch hier ist für die restliche Zeit der Profilgenerierung die Sollgeschwindigkeit zu hoch, als dass der Schrittmotor der Sollbewegung angemessen folgen könnte. Nach dem das Ende des Sollprofils erreicht ist wird der Schrittmotor jetzt durch den Lageregler mit einer geringen Arbeitsfrequenz zum Ziel geführt, der er ohne Rampe folgen kann. Dabei erzeugt er ein recht hohes Moment. Allerdings ist der Zeitbedarf für diese Korrekturmaßnahme sehr hoch.

Erkennung und Handhabung von Stall-Situationen unter Verwendung eines Encoders

Um geeignete Gegenmaßnahmen ergreifen zu können ist zunächst ein Erkennen des Problems nötig. Das nachfolgende Bild ergibt sich, wenn ein *MC_PowerStepper* Baustein verwendet wird. In seiner Parameterstruktur vom Typ ST_PowerStepperStruct [▶ 115] ist als DestallDetectMode PwStDetectMode_Lagging eingetragen. Der Baustein verwendet als Entscheidungsgrundlage den Schleppabstand der Achse und verwendet dabei den Grenzwert und die Filterzeit der hier zu deaktivierenden Schleppüberwachung aus den NC-Achsdaten. In diesem Beispiel ist PwStMode_SetError als DestallMode eingetragen. Das Ergebnis ist zunächst nur durch den anderen Fehlercode vom Schleppabstands-Alarm zu unterscheiden.



Wenn PwStMode_UseOverride als DestallMode eingetragen ist verwendet der *MC_PowerStepper* Baustein den Override, um das Profil unmittelbar zu stoppen. Da dieses Anhalten jedoch keinen Abbruch des Profils bewirkt, gleichzeitig aber das Erreichen des Profilendes verhindert werden keine Statusbits beeinflusst. Die hier auf 2% begrenzte Reglerausgabe führt die Achse bis auf die Schleppabstandsgrenze an die aktuelle Sollposition des Profils heran. Dann wird der Override wieder auf den von der Applikation vorgegebenen Wert gesetzt.



Im Ergebnis wird ein erheblich größerer Teil des Gesamt-Profils mit der vorgegebenen Geschwindigkeit gefahren und die Zielposition korrekt erreicht. Die Statusinformationen der Achse werden korrekt erzeugt.

Kombinationen von Stall-Erkennung und -Handhabung

Die nachfolgende Tabelle stellt Kombinationen der unterstützten Modi zur Stall-Erkennung und -Handhabung dar.

	PwStMode_SetError	PwStMode_SetErrNonRef	PwStMode_UseOverride
PwStDetectMode_Encoderless	Anmerkung 1	Anmerkung 2	nicht geeignet
PwStDetectMode_Lagging	Anmerkung 3	nicht sinnvoll	Anmerkung 4

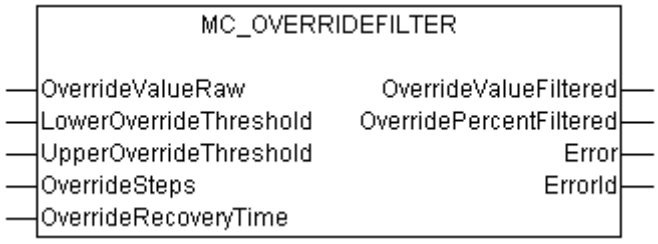
Anmerkung 1: Sinnvoll bei Achsen ohne Encoder, die nicht referenziert werden.

Anmerkung 2: Sinnvoll bei Achsen ohne Encoder, die unter Verwendung des Pulszählers der Klemme und z. B. eines externen Sensors referenziert werden.

Anmerkung 3: Das erzeugte Verhalten entspricht weitgehend der Schleppüberwachung.

Anmerkung 4: Sinnvoll bei Achsen mit Encoder.

6.5.3 MC_OverrideFilter



Mit dem Funktionsbaustein *MC_OverrideFilter* kann ein ungefilterter Overridewert in Digits (z. B. ein Spannungswert einer Analogeingangsklemme) in einen gefilterten, für das zyklische Achsinterface (PlcToNc) passenden, Overidewert (DWORD im Wertebereich 0...1000000) umgerechnet werden. Ebenfalls steht dieser gefilterte Override auch in Prozent zur Verfügung (LREAL im Wertebereich 0...100%).
Der rohe Eingangswert wird dabei durch *LowerOverrideThreshold* und *UpperOverrideThresholdauf* auf einen Gültigkeitsbereich begrenzt und in eine parametrierbare Stufung (Auflösung) umgesetzt (*OverrideSteps*).
Nach jeder Overrideänderung am Ausgang des FB's wird intern eine Mindestruhezeit abgewartet (*OverrideRecoveryTime*), bevor wieder ein neuer Overridewert übernommen werden kann. Die einzigen Ausnahmen stellen die Overridewerte 0% und 100%, die aus Sicherheitsgründen immer ohne Zeitverzögerung umgesetzt werden.

Aufgrund der Stufung des ausgegebenen Override-Wertes (*OverrideValueFiltered*) kann es bei sehr kleinen Override-Eingangswerten (*OverrideValueRaw*) vorkommen, dass der gefilterte Override zu Null wird. Ein Override Null führt zum Stillstand der Achse. Falls ein vollständiger Stillstand nicht erwünscht ist, sollte *OverrideValueRaw* nicht unter die kleinste Stufung fallen.

Eingänge

```
VAR_INPUT
OverrideValueRaw       : DINT;
LowerOverrideThreshold : DINT := 0;     (* 0...32767 digits *)
UpperOverrideThreshold : DINT := 32767;(* 0...32767 digits *)
OverrideSteps          : UDINT := 200; (* 200 steps=> 0.5 percent*)
OverrideRecoveryTime   : TIME := T#150ms; (* 150 ms *)
END_VAR
```

OverrideValueRaw	Rohe, ungefilterter Overridewert (z.B. ein Spannungswert einer Analogeingangsklemme).
LowerOverrideThreshold	Die untere Schranke, die den rohen Overridewert begrenzt.
UpperOverrideThreshold	Die obere Schranke, die den rohen Overridewert begrenzt.
OverrideSteps	Die vorgegebene Stufung (Overrideauflösung).
OverrideRecoveryTime	Mindestruhezeit nach der ein neuer gefilterter Overridewert auf den Ausgang gelegt wird. Die Overridewerte 0% und 100% werden allerdings ohne Zeitverzögerung umgesetzt.

Ausgänge

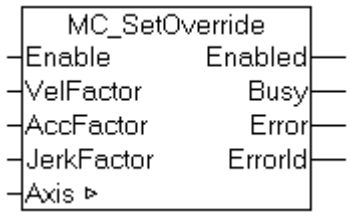
```
VAR_OUTPUT
OverrideValueFiltered   : DWORD; (* 0...1000000 counts *)
OverridePercentFiltered : LREAL; (* 0...100 % *)
Error                   : BOOL;
ErrorId                 : UDINT;
END_VAR
```

OverrideValueFiltered	Der gefilterte Overridewert in Digits (der Datentyp passt zum Override im zyklischen Achsinterface 0..1000000).

OverridePercentFiltered	Der gefilterte Overridewert in Prozent (0..100%).
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Mögliche Fehlernummer	Mögliche Ursachen
MC_ERROR_PARAMETER_NOT_CORRECT	• OverrideSteps <= 1 • LowerOverrideThreshold >= UpperOverrideThreshold

6.5.4 MC_SetOverride



Mit dem Funktionsbaustein *MC_SetOverride* kann der Override einer Achse vorgegeben werden.

Eingänge

```
VAR_INPUT
Enable     : BOOL; (* B *)
VelFactor  : LREAL (* B *) := 1.0; (*1.0 = 100%*)
AccFactor  : LREAL (* E *) := 1.0; (*1.0 = 100%*)(* not supported *)
JerkFactor : LREAL (* E *) := 1.0; (*1.0 = 100%*)(* not supported *)
END_VAR
```

Enable	Solange *Enable* aktiv ist, wird das Kommando ausgeführt.
VelFactor	Geschwindigkeits-Override-Faktor
AccFactor	nicht unterstützt
JerkFactor	nicht unterstützt

Ausgänge

```
VAR_OUTPUT
Enabled : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Enabled	Der parametrierte Override wird gesetzt
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Enable* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

```
VAR_IN_OUT
Axis   : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.5.5 MC_SetEncoderScalingFactor

MC_SetEncoderScalingFactor ändert den Skalierungsfaktor des aktiven Encoders einer Achse entweder im Stillstand oder in Bewegung.

Die Änderung kann absolut oder relativ durchgeführt werden. Da sich durch Änderung des Skalierungsfaktors im absoluten Modus ein Positionssprung ergibt, ist dieser Mode nur im Stilstand geeignet. Im relativen Modus wird gleichzeitig ein interner Positionsoffset so angepasst, dass es keinen Sprung gibt. Es ist aber zu beachten, dass sich durch einen Eingriff während der Fahrt bei gleichbleibender realer Geschwindigkeit die Istgeschwindigkeit der Achse ändert. Daher können nur kleine Änderungen während der Fahrt durchgeführt werden.

Eingänge

```
VAR_INPUT
Execute       : BOOL;
ScalingFactor : LREAL;
Mode          : E_SetScalingFactorMode;
Options : ST_SetEncoderScalingOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.	
ScalingFactor	Skalierungsfaktor des aktiven Encoders einer Achse. Der Skalierungsfaktor wird in physikalischen Positionseinheiten [u] dividiert durch die Anzahl der Encoder-Inkremente angegeben.	
Mode	Der Skalierungsfaktor kann im absoluten oder relativen Modus gesetzt werden (*ENCODERSCALINGMODE_ABSOLUTE, ENCODERSCALINGMODE_RELATIVE*). Im absoluten Modus beginnt die Zählung im Nullpunkt des Achskoordinatensystems, daher ergibt sich ein Positionssprung, wenn der Skalierungsfaktor geändert wird. Im relativen Modus ändert sich die Istposition der Achse nicht. Dieser *Mode* ist daher auch für die Änderung in Bewegung geeignet.	
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.	
Options.	**SelectEncoderIndex**	SelectEncoderIndex kann optional gesetzt werden, falls eine Achse mit mehreren Encodern verwendet wird und die Position eines bestimmten Encoders (*Options.EncoderIndex*) gesetzt werden soll.
Options.	**EncoderIndex**	EncoderIndex gibt den Encoder (0..n) an falls *SelectEncoderIndex* TRUE ist.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
Options : ST_SetPositionOptions;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE wenn die Position erfolgreich gesetzt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer:

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

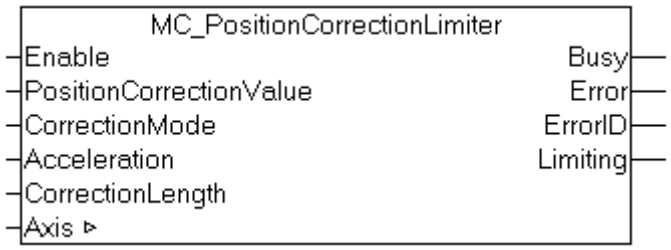
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.5.6 MC_PositionCorrectionLimiter



Der Funktionsbaustein *MC_PositionCorrectionLimiter* schreibt einen Korrekturwert (*PositionCorrectionValue*) auf die Istposition einer Achse. Abhängig vom Korrekturmodus werden die Daten entweder unmittelbar oder gefiltert in die Achse eingespeist.

VAR_INPUT

```
VAR_INPUT
Enable                  : BOOL;
PositionCorrectionValue : LREAL;
CorrectionMode          : E_AxisPositionCorrectionMode;
Acceleration            : LREAL;
CorrectionLenght        : LREAL;
END_VAR
```

Enable	Das kontinuierliche Schreiben des *PositionCorrectionValue* wird durch diesen Eingang aktiviert. Er muss TRUE sein, solange neue Korrekturwerte akzeptiert werden sollen.
PositionCorrectionValue	Der Korrekturwert, der zum Istwert der Achse addiert werden soll.
CorrectionMode	Abhängig von diesem Modus wird der *PositionCorrectionValue* entweder unmittelbar oder gefiltert geschrieben. Für eine ausführliche Beschreibung siehe E_AxisPositionCorrectionMode [▶ 117].
Acceleration	Abhängig vom *CorrectionMode* wird hier die maximale Beschleunigung zum Erreichen des neuen Korrekturwerts vorgegeben. Bei PositionCorrectionMode_Fast [▶ 117] hat dieser Wert einen unmittelbaren Einfluss auf das Positionsdelta per PLC-Tick. Max. zulässiger Korrekturwert Positionsdelta = Beschleunigung * (SPS-Zykluszeit)^2. Falls Acceleration gleich 0.0 parametriert wird, wird die Positionskorrektur nicht begrenzt.
CorrectionLength	Wenn der *CorrectionMode* mit PositionCorrectionMode_FullLength [▶ 117] übereinstimmt, wird dieser Parameter aktiv. Eine Änderung beim *PositonCorrectionValue* wird auf dieser Korrekturlänge aufgeteilt.

VAR_IN_OUT

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur AXIS_REF [▶ 105]

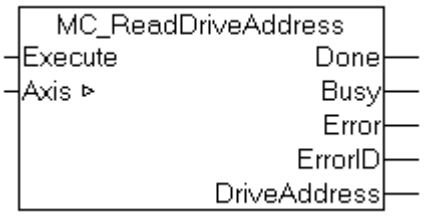
VAR_OUTPUT

```
VAR_OUTPUT
Busy      : LREAL;
Error     : BOOL;
ErrorId   : UDINT;
Limiting : BOOL;
ND_VAR
```

Busy	Wird TRUE, sobald der Funktionsbaustein aktiv ist und FALSE, wenn er in den ursprünglichen Zustand zurückgekehrt.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorId	Wenn der Fehlerausgang gesetzt ist, liefert dieser Parameter die Fehlernummer.
Limiting	Wird TRUE, wenn der geforderte *PositionCorrectionValue* noch nicht vollständig akzeptiert ist.

Um diesen Funktionsbaustein erfolgreich zu nutzen, muss der Parameter *Positionskorrektur* im System Manager aktiviert werden.

6.5.7 MC_ReadDriveAddress



MC_ReadDriveAddress liest die ADS Zugriffdaten für ein an die Achse angeschlossenes Antriebsgerät. Diese Informationen werden für den Zugriff auf das Gerät, zum Beispiel zur speziellen Parametrierung, benötigt.

Eingänge

```
VAR_INPUT
Execute : BOOL; (* B *)
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done         : BOOL;   (* B *)
Busy         : BOOL;   (* E *)
Error        : BOOL;   (* B *)
ErrorID      : DWORD;  (* B *)
DriveAddress : ST_DriveAddress; (* B *)

END_VAR
```

Done	Wird TRUE wenn das Kommando fehlerfrei ausgeführt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.
DriveAddress	ADS Zugriffsdaten [▶ 115] eines mit der Achse verbunden Antriebsgerätes.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

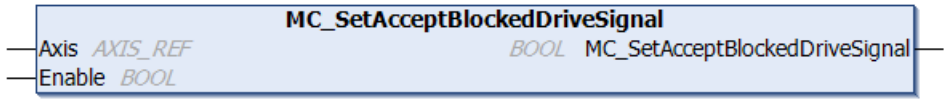
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

6.5.8 MC_SetAcceptBlockedDriveSignal



Es gibt Situationen, in denen ein Antrieb nicht mehr den NC-Sollwerten folgt, beispielsweise wenn eine Achse auf einen Endschalter fährt. Eine solche Situation wird von der NC als Fehler erkannt und der Antrieb wird stillgesetzt. Der Anwender kann aber auch bewusst eine solche Situation provozieren wollen, z. B. um für einen Referenzierfahrt auf einen Endschalter zu fahren. Mit der Funktion MC_SetAcceptBlockedDriveSignal kann der Anwender vorrübergehend verhindern, dass die NC-Achse in einen Fehler läuft, weil der Antrieb den Sollwerten der NC nicht mehr folgt.

- Siehe auch Bit 8 des ControlDWords im AXIS_REF.
- Ein SERCOS/SoE-Antrieb meldet über das Statusbit 3 des Antriebsstatuswortes S-0-0135 "Drive follows the command values".
- Ein CanOpen/CoE-Antrieb meldet über das Statusbit 12 des Objekts 6041h "Drive follows the command values".

FUNCTION MC_SetAcceptBlockedDriveSignal: BOOL

Eingänge

```
VAR_INPUT
    Enable :  BOOL;
END_VAR
```

Enable: NC-Reglerfreigabe für die Achse

Ein-/Ausgänge

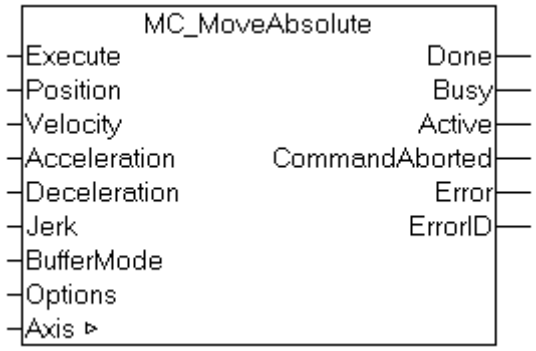
```
VAR_IN_OUT
    Axis : AXIS_REF;
END_VAR
```

Axis: Achsdatenstruktur, die eine Achse eindeutig im System adressiert. Sie enthält unter anderem den aktuellen Status der Achse wie Position, Geschwindigkeit oder Fehlerzustand. (Typ: AXIS_REF [▶ 105])

7 Motion-Funktionsbausteine

7.1 Point to Point Motion

7.1.1 MC_MoveAbsolute



MC_MoveAbsolute startet eine Positionierung auf eine absolute Zielposition und überwacht die Achsbewegung über den gesamten Fahrweg. Der Done-Ausgang wird gesetzt, wenn die Zielposition angefahren wurde. Anderenfalls wird der CommandAborted- oder im Fehlerfall der Error-Ausgang gesetzt.

MC_MoveAbsolute wird in erster Linie für lineare Achssysteme eingesetzt. Bei Modulo-Achsen wird die Position nicht als Modulo-Position, sondern ebenfalls als absolute Position in einem endlosen absoluten Koordinatensystem interpretiert. Zur Modulo-Positionierung kann alternativ der Baustein MC_MoveModulo [▶ 66] verwendet werden.

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_MoveAbsolute* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.

Eingänge

```
VAR_INPUT
Execute      : BOOL;
Position     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk         : LREAL;
BufferMode   : MC_BufferMode;
Options      : ST_MoveOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Position	Absolute Zielposition auf die positioniert werden soll.
Velocity	Maximale Geschwindigkeit mit der gefahren werden soll (>0).
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.

Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Wird das Kommando auf eine gekoppelte Slave-Achse angewendet, so ist nur der Buffer-Mode *Aborting* möglich. Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done            : BOOL;
Busy            : BOOL;
Active          : BOOL;
CommandAborted : BOOL;
Error           : BOOL;
ErrorID         : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Zielposition erreicht wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

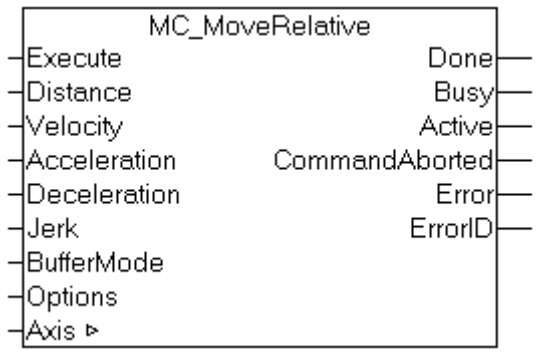
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.1.2 MC_MoveRelative



MC_MoveRelative startet eine relative Positionierung ausgehend von der aktuellen Sollposition und überwacht die Achsbewegung über den gesamten Fahrweg. Der Done-Ausgang wird gesetzt, wenn die Zielposition angefahren wurde. Anderenfalls wird der *CommandAborted*- oder im Fehlerfall der *Error*-Ausgang gesetzt.

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_MoveAbsolute* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.

Eingänge

```
VAR_INPUT
Execute      : BOOL;
Distance     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk         : LREAL;
BufferMode   : MC_BufferMode;
Options      : ST_MoveOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Distance	Relative Fahrstrecke um die positioniert werden soll.
Velocity	Maximale Geschwindigkeit mit der gefahren werden soll (>0).
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.

Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Wird das Kommando auf eine gekoppelte Slave-Achse angewendet, so ist nur der Buffer-Mode *Aborting* möglich. Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done            : BOOL;
Busy            : BOOL;
Active          : BOOL;
CommandAborted : BOOL;
Error           : BOOL;
ErrorID         : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Zielposition erreicht wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

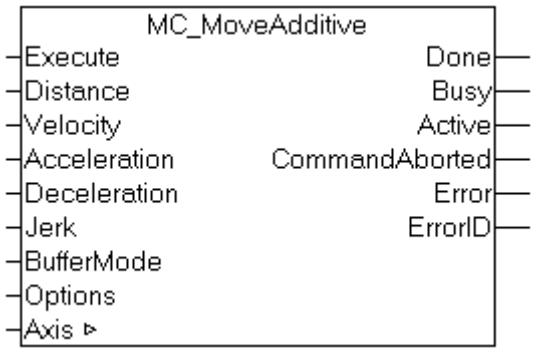
Ein/Ausgänge

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.1.3 MC_MoveAdditive



MC_MoveAdditive startet eine relative Positionierung ausgehend von der letzten beauftragten Zielposition unabhängig davon ob diese erreicht wurde. Der Done-Ausgang wird gesetzt, wenn die Zielposition angefahren wurde. Anderenfalls wird der *CommandAborted*- oder im Fehlerfall der *Error*-Ausgang gesetzt.

Ist keine letzte Zielposition bekannt oder bewegt sich die Achse endlos, so wird die Bewegung ausgehend von der aktuellen Sollposition der Achse ausgeführt.

Eingänge

```
VAR_INPUT
Execute      : BOOL;
Distance     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk         : LREAL;
BufferMode   : MC_BufferMode;
Options      : ST_MoveOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Distance	Relative Fahrstrecke um die positioniert werden soll.
Velocity	Maximale Geschwindigkeit mit der gefahren werden soll (>0).
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.

BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done            : BOOL;
Busy            : BOOL;
Active          : BOOL;
CommandAborted : BOOL;
Error           : BOOL;
ErrorID         : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Zielposition erreicht wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

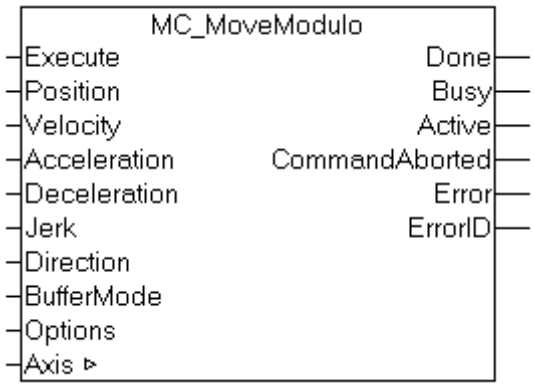
```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

MC_MoveAdditive ist nicht implementiert für Eil-/Schleichachsen.

7.1.4 MC_MoveModulo



Mit dem Funktionsbaustein *MC_MoveModulo* wird eine Positionierung durchgeführt, die sich auf die Moduloposition einer Achse bezieht. Grundlage für eine Moduloumdrehung ist dabei der einstellbare Achsparameter *Modulofaktor* (z. B. 360°). Abhängig vom *Direction* Eingang werden drei mögliche Starttypen unterschieden.

- Positionierung in *positive Richtung*
- Positionierung in *negative Richtung*
- Positionierung *auf kürzestem Weg*

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_MoveModulo* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.

Start einer Achse aus dem Stillstand

Wird eine Achse mit *MC_MoveModulo* aus dem Stillstand gestartet, so können auch Positionen größer oder gleich 360° angegeben werden, um zusätzliche volle Umdrehungen auszuführen. Gleiches gilt für einen Start mit dem *BufferModeMC_Buffered*.

Start einer Achse aus der Bewegung

Falls sich eine Achse bereits in Bewegung befindet, gibt es einige Besonderheiten zu beachten. Die Bewegungsrichtung kann durch MC_MoveModulo nicht umgekehrt werden, sondern führt nur in der aktuellen Richtung zum Ziel. Weiterhin kann die Anzahl zusätzlicher Umdrehungen nicht vom Anwender bestimmt werden. Das System rechnet selbständig, wie die Achse auf möglichst kurzem Weg auf die Zielposition positioniert werden kann.

Der Fehlerausgang muss zwingend ausgewertet werden, da unter bestimmten Bedingungen ein orientierter Stopp nicht möglich ist. Beispielsweise könnte kurz zuvor ein Standardstopp ausgelöst worden sein oder es würde im Falle eines orientierten Stopps ein aktiver Software-Endschalter überfahren. In allen Fehlerfällen wird die Achse sicher gestoppt, steht dann aber anschließend nicht an der gewünschten orientierten Position.

Sonderfälle

Besonders zu beachten ist das Verhalten bei Anforderung einer oder mehrerer vollständiger Moduloumdrehungen. Befindet sich die Achse auf einer exakten Sollposition von beispielsweise 90 Grad und wird sie auf 90 Grad positioniert, so wird keine Bewegung ausgeführt. Bei Anforderung von 450 Grad in positiver Richtung fährt sie eine Umdrehung. Nach einem Achsreset kann das Verhalten anders sein, weil

durch den Reset die aktuelle Istposition in die Sollposition übernommen wird. Damit steht die Achse nicht mehr exakt bei 90 Grad, sondern ein wenig darunter oder darüber. In diesen beiden Fällen wird entweder nur eine minimale Positionierung auf 90 Grad oder aber eine ganze Umdrehung ausgeführt. Auf dieses Problematik wird in den Erläuterungen [▶ 69] näher eingegangen.

Je nach Anwendungsfall kann es für volle Moduloumdrehungen günstiger sein, die gewünschte Zielposition auf Grund der aktuellen absoluten Position zu berechnen und mit dem Baustein MC_MoveAbsolute [▶ 60] zu positionieren.

Die Modulopositionierung ebenso wie die Absolutpositionierung steht für alle Achsen unabhängig von der *Modulo* Einstellung im TwinCAT SystemManager zur Verfügung. Für jede Achse kann die aktuelle Absolutposition *SetPos* aus dem zyklischen Achsinterface Datentyp NCTOPLC_AXIS_REF [▶ 106] ausgelesen werden.

Wichtig: Nähere Erläuterungen zu Modulo Bewegungen [▶ 69]

Eingänge

```
VAR_INPUT
Execute : BOOL;
Position : LREAL;
Velocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
Direction : MC_Direction;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 107] MC_Direction [▶ 109]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Position	Modulo-Zielposition auf die positioniert werden soll. Falls die Achse aus dem Stillstand gestartet wird, führen Positionen ab 360° zu zusätzlichen Umdrehungen. Negative Positionen sind nicht zulässig.
Velocity	Maximale Geschwindigkeit mit der gefahren werden soll (>0).
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.
Direction	Positive oder negative Fahrtrichtung vom Typ MC_Direction [▶ 109]. Falls die Achse aus der Bewegung heraus gestartet wird, darf die Richtung nicht umgekehrt werden.
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Um den BufferMode zu verwenden, ist immer ein

	zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Zielposition erreicht wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

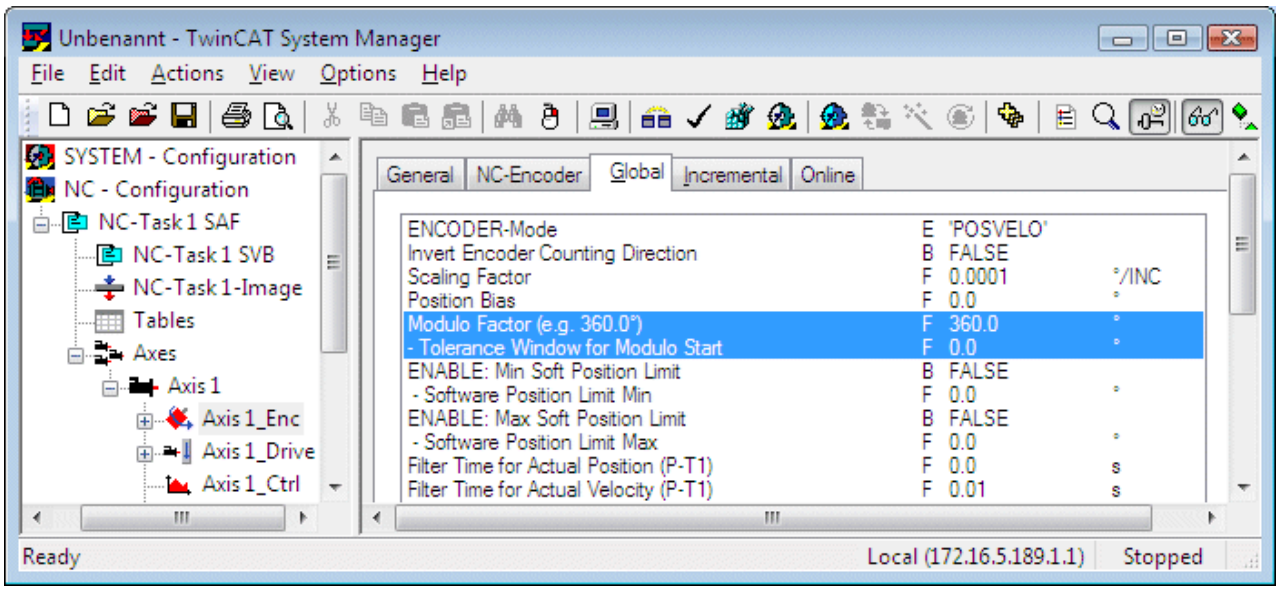
Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.1.5 Hinweise zur Modulo-Positionierung

Die Modulo-Positionierung (MC_MoveModulo [▶ 66]) ist unabhängig vom Achstyp möglich. Sie kann also bei linearen ebenso wie bei rotatorischen Achsen angewendet werden, da TwinCAT nicht zwischen diesen Typen unterscheidet. Auch eine Modulo-Achse hat eine fortlaufende absolute Position im Bereich ±∞. Die Modulo-Position der Achse ist einfach eine zusätzliche Information zur absoluten Achsposition und die Modulo-Positionierung stellt die gewünschte Zielposition auf eine andere Art dar. Im Gegensatz zur absoluten Positionierung, bei der der Benutzer das Ziel eindeutig vorgibt, birgt die Modulo-Positionierung einige Tücken, da die gewünschte Zielposition unterschiedlich interpretiert werden kann.

Einstellungen im TwinCAT SystemManager

Die Modulo-Positionierung bezieht sich grundsätzlich auf eine im TwinCAT SystemManager einstellbare *Modulo-Periode*. In den Beispielen auf dieser Seite wird von einer rotatorischen Achse mit einer *Modulo-Periode* von 360 Grad ausgegangen.



Das *Modulo-Toleranzfenster* definiert ein Positionsfenster um die aktuelle Modulo-Sollposition der Achse herum. Die Fensterbreite entspricht dem doppelten angegebenen Wert (Sollposition ± Toleranzwert). Auf das Toleranzfenster wird im Folgenden näher eingegangen.

Besonderheiten beim Reset einer Achse

Die Positionierung einer Achse bezieht sich immer auf deren Sollposition. Die Sollposition der Achse ist im Normalfall die Position, die mit dem letzten Fahrauftrag angefahren wurde. Durch einen Achsreset (MC_Reset [▶ 22], Zuschalten der Reglerfreigabe mit MC_Power [▶ 21]) kann sich eine vom Anwender nicht erwartete Sollposition einstellen, da in diesem Fall die aktuelle Istposition als Sollposition übernommen wird. Der Achsreset setzt durch diesen Vorgang einen eventuell aufgetretenen Schleppfehler zurück. Wenn dieser Umstand nicht berücksichtigt wird, kann sich eine nachfolgende Positionierung unerwartet verhalten.

Beispiel: Eine Achse wird auf 90° positioniert, wodurch die Sollposition der Achse anschließend exakt 90° beträgt. Ein weiterer Modulo-Fahrauftrag auf 450° in *positive Richtung* führt zu einer vollen Umdrehung und die Modulo-Position der Achse ist anschließend wieder exakt 90°. Wird jetzt ein Achsreset durchgeführt, so kann die Sollposition zufällig etwas kleiner oder etwas größer als 90° sein. Der neue Wert ist abhängig vom Istwert der Achse zum Zeitpunkt des Reset. In beiden Fällen verhält sich das nächste Fahrkommando unterschiedlich. Liegt die Sollposition leicht unter 90°, so führt ein neues Fahrkommando auf 90° in *positive Richtung* nur zu einer minimalen Bewegung. Die durch den Reset entstandene Abweichung wird ausgeglichen und die Sollposition ist anschließend wieder exakt 90°. Liegt aber die Sollposition nach dem Achsreset leicht über 90°, so führt dasselbe Fahrkommando zu einer vollen Umdrehung um wieder die exakte Sollposition von 90° zu erreichen. Diese Problematik tritt auf, wenn volle Umdrehungen um 360° oder ein Vielfaches von 360° beauftragt werden. Bei Positionierungen auf einen von der aktuellen Modulo-Position entfernten Winkel ist der Fahrauftrag eindeutig.

Um das Problem zu lösen, kann ein *Modulo-Toleranzfenster* im TwinCAT SystemManager parametriert werden. Kleine Abweichungen der Position, die innerhalb des Fensters liegen, führen damit nicht mehr zu einem unterschiedlichen Verhalten der Achse. Wird beispielsweise ein Fenster von 1° parametriert, so verhält sich die Achse im oben beschriebenen Fall gleich, solange die Sollposition zwischen 89° und 91° liegt. Wenn jetzt die Sollposition weniger als 1° über 90° liegt, wird die Achse bei einem Modulo-Start in *positive Richtung* zurückpositioniert. Bei einer Zielposition von 90° wird also in beiden Fällen eine Minimalbewegung auf exakt 90° ausgeführt und bei einer Zielposition von 450° wird in beiden Fällen eine ganze Umdrehung gefahren.

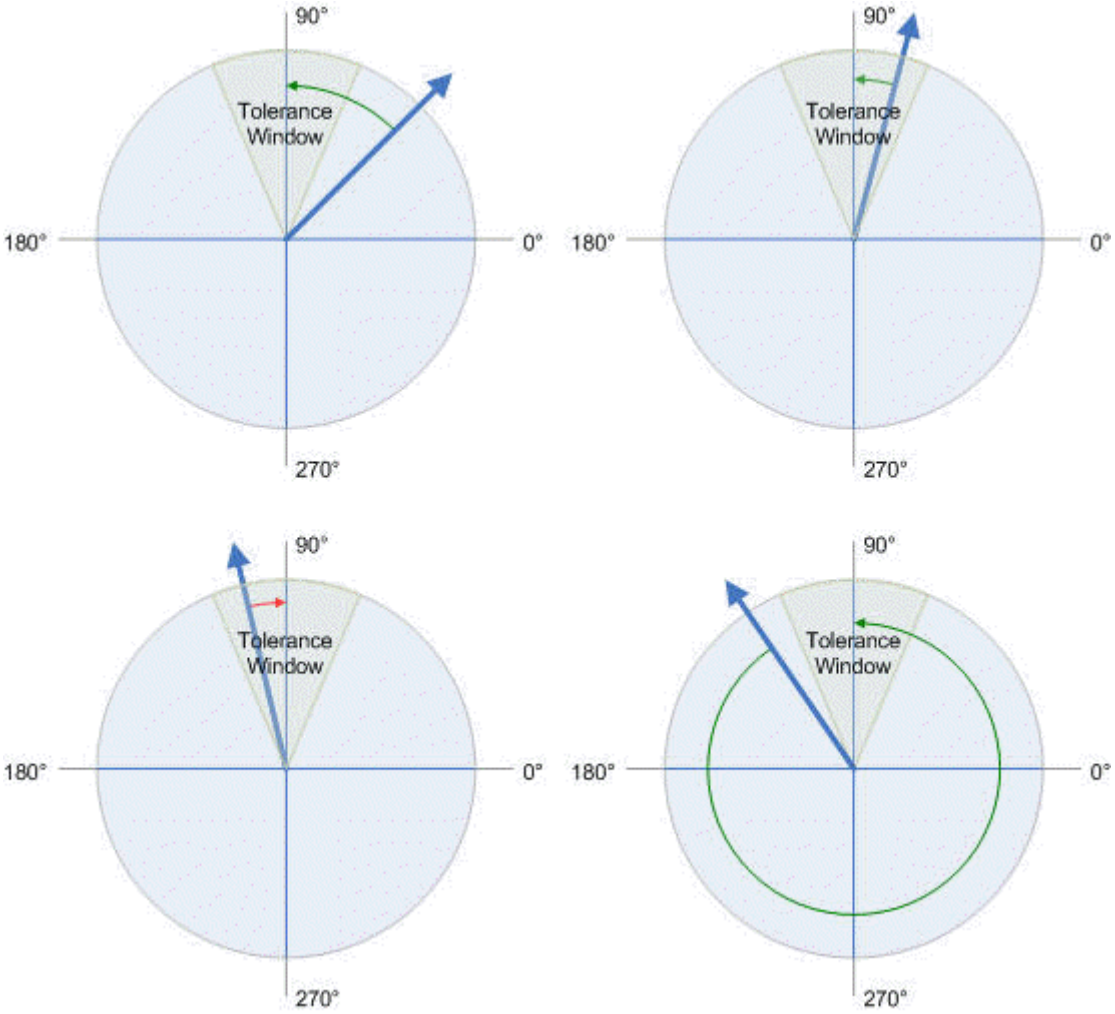


Abbildung: Wirkung des Modulo-Toleranzfensters - Modulo-Zielposition 90° in positive Richtung

Das Modulo-Toleranzfenster kann also innerhalb des Fensters zu Bewegungen gegen die beauftragte Richtung führen. Bei einem kleinen Fenster ist das normalerweise unproblematisch, weil auch Regelabweichungen zwischen Soll- und Istposition in beide Richtungen ausgeglichen werden. Das Toleranzfenster lässt sich also auch bei Achsen verwenden, die konstruktionsbedingt nur in einer Richtung verfahren werden dürfen.

Modulo-Positionierung um weniger als eine Umdrehung

Die Modulo-Positionierung von einer Ausgangsposition auf eine nicht identische Zielposition ist eindeutig und birgt keine Besonderheiten. Eine Modulo-Zielposition im Bereich [0 ≤ Position < 360] führt in weniger als einer ganzen Umdrehung zum gewünschten Ziel. Ist die Zielposition mit der Ausgangsposition identisch, so wird keine Bewegung ausgeführt. Bei Zielpositionen ab 360 Grad aufwärts werden ein oder mehr vollständige Umdrehungen ausgeführt, bevor die Achse auf die gewünschte Zielposition fährt.

Für eine Bewegung von 270° auf 0° darf demnach nicht 360°, sondern es muss 0° als Modulo-Zielposition abgegeben werden, da 360 außerhalb des Grundbereiches liegt und zu einer zusätzlichen Umdrehung führen würde.

Die Modulo-Positionierung unterscheidet drei Richtungsvorgaben, *positive Richtung*, *negative Richtung* und *auf kürzestem Weg* (MC_Direction [▶ 109]). Bei der Positionierung auf kürzestem Weg sind Zielpositionen ab 360° nicht sinnvoll, da das Ziel immer direkt angefahren wird. Im Gegensatz zur positiven oder negativen Richtung können also nicht mehrere Umdrehungen ausgeführt werden, bevor das Ziel angefahren wird.

Wichtig: Bei Modulo-Positionierungen mit dem Start-Typ *auf kürzestem Weg* sind nur Modulo-Zielpositionen in der Grundperiode (z. B. kleiner als 360°) erlaubt, anderenfalls wird ein Fehler zurückgegeben.

Die folgende Tabelle zeigt einige Positionierungsbeispiele:

Richtung (Modulo-Starttyp)	Absolute Anfangsposition	Modulo-Zielposition	Relativer Verfahrweg	absolute Endposition	Modulo Endposition
positive Richtung	90,00	0,00	270,00	360,00	0,00
positive Richtung	90,00	360,00	630,00	720,00	0,00
positive Richtung	90,00	720,00	990,00	1080,00	0,00
negative Richtung	90,00	0,00	-90,00	0,00	0,00
negative Richtung	90,00	360,00	-450,00	-360,00	0,00
negative Richtung	90,00	720,00	-810,00	-720,00	0,00
auf kürzestem Weg	90,00	0,00	-90,00	0,00	0,00

Modulo-Positionierung um ganze Umdrehungen

Modulo-Positionierungen um ein oder mehrere ganze Umdrehungen verhalten sich grundsätzlich nicht anders als Positionierungen auf von der Ausgangsposition entfernt liegende Winkel. Wenn die beauftragte Zielposition gleich der Ausgangsposition ist, so wird keine Bewegung ausgeführt. Für eine ganze Umdrehung muss zur Ausgangsposition 360° addiert werden.

Das weiter oben beschriebene Reset-Verhalten zeigt , dass Positionierungen mit ganzzahligen Umdrehungen besonders beachtet werden müssen. Die nachfolgende Tabelle zeigt Positionierbeispiele für eine Ausgangsposition von ungefähr 90°. Das Modulo-Toleranzfenster TF ist hier auf 1° eingestellt. Besondere Fälle, in denen die Ausgangsposition außerhalb dieses Fensters liegt, sind gekennzeichnet.

Richtung (Modulo-Starttyp)	Absolute Anfangsposition	Modulo-Zielposition	Relativer Verfahrweg	absolute Endposition	Modulo Endposition	Anmerkung
positive Richtung	90,00	90,00	0,00	90,00	90,00	
positive Richtung	90,90	90,00	-0,90	90,00	90,00	
positive Richtung	91,10	90,00	358,90	450,00	90,00	außerhalb TF
positive Richtung	89,10	90,00	0,90	90,00	90,00	
positive Richtung	88,90	90,00	1,10	90,00	90,00	außerhalb TF
positive Richtung	90,00	450,00	360,00	450,00	90,00	
positive Richtung	90,90	450,00	359,10	450,00	90,00	

positive Richtung	91,10	450,00	718,90	810,00	90,00 außerhalb TF
positive Richtung	89,10	450,00	360,90	450,00	90,00
positive Richtung	88,90	450,00	361,10	450,00	90,00 außerhalb TF
positive Richtung	90,00	810,00	720,00	810,00	90,00
positive Richtung	90,90	810,00	719,10	810,00	90,00
positive Richtung	91,10	810,00	1078,90	1170,00	90,00 außerhalb TF
positive Richtung	89,10	810,00	720,90	810,00	90,00
positive Richtung	88,90	810,00	721,10	810,00	90,00 außerhalb TF
negative Richtung	90,00	90,00	0,00	90,00	90,00
negative Richtung	90,90	90,00	-0,90	90,00	90,00
negative Richtung	91,10	90,00	-1,10	90,00	90,00 außerhalb TF
negative Richtung	89,10	90,00	0,90	90,00	90,00
negative Richtung	88,90	90,00	-358,90	-270,00	90,00 außerhalb TF
negative Richtung	90,00	450,00	-360,00	-270,00	90,00
negative Richtung	90,90	450,00	-360,90	-270,00	90,00
negative Richtung	91,10	450,00	-361,10	-270,00	90,00 außerhalb TF
negative Richtung	89,10	450,00	-359,10	-270,00	90,00
negative Richtung	88,90	450,00	-718,90	-630,00	90,00 außerhalb TF
negative Richtung	90,00	810,00	-720,00	-630,00	90,00
negative Richtung	90,90	810,00	-720,90	-630,00	90,00
negative Richtung	91,10	810,00	-721,10	-630,00	90,00 außerhalb TF
negative Richtung	89,10	810,00	-719,10	-630,00	90,00
negative Richtung	88,90	810,00	-1078,90	-990,00	90,00 außerhalb TF

Modulo-Berechnungen im SPS Programm

Alle Positionieraufträge an eine Achse werden in TwinCAT NC auf der Basis der *Sollposition* durchgeführt. Die aktuelle Istposition wird nur zur Regelung herangezogen. Wenn in einem SPS-Programm eine neue Zielposition ausgehend von der aktuellen Position berechnet werden soll, so muss diese Berechnung mit der aktuellen Sollposition der Achse durchgeführt werden (Axis.NcToPlc.ModuloSetPos und Axis.NcToPlc.ModuloSetTurns).

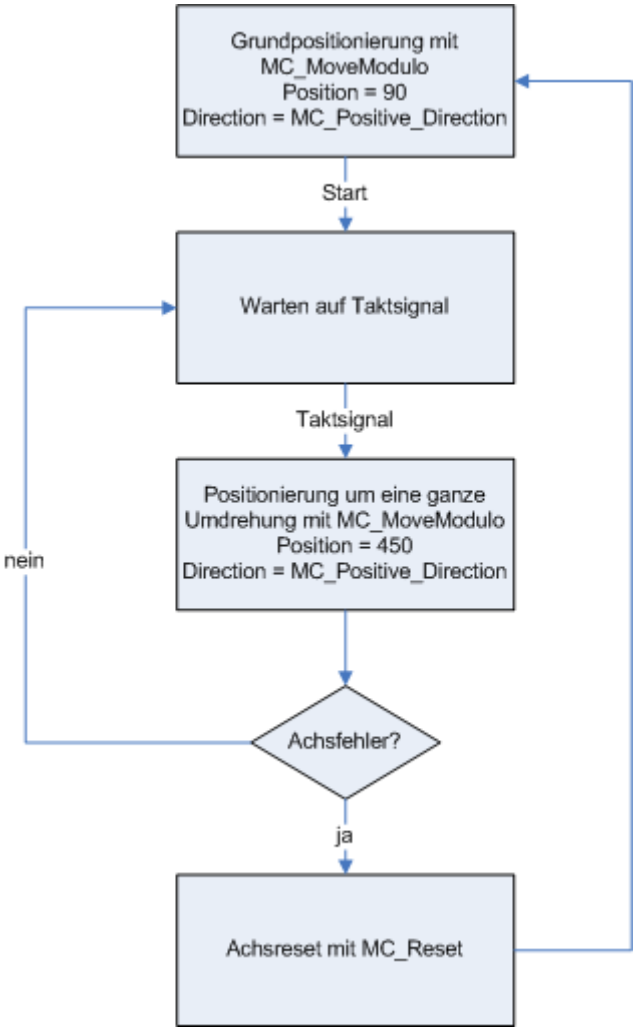
Es ist nicht zu empfehlen, Auftragsberechnungen auf Basis der Modulo-Istposition durchzuführen, die im zyklischen Achsinterface (*ModuloActPos* und *ModuloActTurns*) zur Verfügung steht. Wegen der mehr oder weniger großen Regelabweichung der Achse könnten sich Fehler im programmierten Ablauf, wie z. B. unerwünschte Umdrehungen, ergeben.

Anwendungsbeispiel

In einer Anlage führt eine Rotationsachse einen Arbeitsschritt aus. Die Ausgangsposition für jeden Arbeitsschritt ist 90° und mit jedem Takt soll die Achse um 360° in positive Richtung positioniert werden. Eine Rückwärtspositionierung ist aus mechanischen Gründen nicht erlaubt. Kleine Rückwärtspositionierungen im Rahmen der Lageregelung sind zulässig.

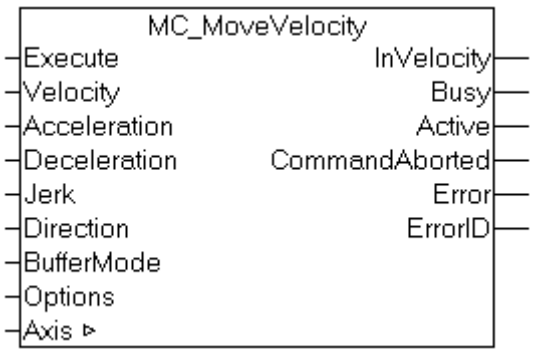
Das Modulo-Toleranzfenster wird im SystemManager auf 1,5° eingestellt. Damit werden unerwünschte Umdrehungen der Achse nach einem Achsreset vermieden. Da die Achse nur vorpositioniert werden darf, wird das Kommando MC_MoveModulo [▶ 66] mit dem Modulo-Starttyp *positive Richtung (MC_Positive_Direction)* verwendet. Die Modulo-Zielposition wird mit 450° angegeben, da die Ausgangsorientierung nach einer vollen Umdrehung um 360° wieder erreicht werden soll. Eine Modulo-Zielposition von 90° würde hier keine Bewegung ausführen.

Der Ablauf startet zunächst mit einer Grundpositionierung (MC_MoveModulo [▶ 66]), mit der die exakte Ausgangsposition sichergestellt wird. Anschließend wechselt die Schrittkette in einen Bearbeitungszyklus. Im Fehlerfall wird die Achse mit MC_Reset [▶ 22] zurückgesetzt und anschließend, am Anfang der Schrittkette, auf ihre gültige Ausgangsposition gefahren. In diesem Fall wird 90° als Zielposition angegeben, damit diese Position schnellst möglich angefahren wird. Steht die Achse bereits an der Ausgangsposition, so wird keine Bewegung ausgeführt.



Der Reset-Schritt kann alternativ auch am Anfang der Schrittkette ausgeführt werden um die Achse auch zu Beginn des Ablaufs zu initialisieren.

7.1.6 MC_MoveVelocity



MC_MoveVelocity startet eine Endlosfahrt mit vorgegebener Geschwindigkeit und Richtung. Die Bewegung kann durch ein Stopp-Kommando angehalten werden.

Der *InVelocity*-Ausgang wird gesetzt, sobald die konstante Geschwindigkeit erreicht ist. Mit Erreichen der Konstantfahrt ist die Funktion des Bausteins abgeschlossen, es findet also keine weitere Überwachung der Bewegung statt. Wenn das Kommando noch während der Beschleunigungsphase abgebrochen wird, wird der CommandAborted- oder im Fehlerfall der Error-Ausgang gesetzt.

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_MoveAbsolute* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.

Eingänge

```
VAR_INPUT
Execute : BOOL; (* B *)
Velocity : LREAL; (* E *)
Acceleration : LREAL; (* E *)
Deceleration : LREAL; (* E *)
Jerk : LREAL; (* E *)
Direction : MC_Direction := MC_Positive_Direction; (* E *)
BufferMode : MC_BufferMode; (* E *)
Options : ST_MoveOptions; (* V *)
END_VAR
```

MC_BufferMode [▶ 107] MC_Direction [▶ 109]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Velocity	Geschwindigkeit mit der gefahren werden soll (>0).
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.
Direction	positive oder negative Fahrtrichtung vom Typ MC_Direction [▶ 109].
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Wird das Kommando auf eine gekoppelte Slave-Achse angewendet, so ist nur der Buffer-Mode *Aborting* möglich. Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
InVelocity : BOOL; (* B *)
Busy : BOOL; (* E *)
Active : BOOL; (* E *)
CommandAborted : BOOL; (* E *)
Error : BOOL; (* B *)
```

```
ErrorID : UDINT; (* E *)
END_VAR
```

InVelocity	Der Ausgang *InVelocity* wird TRUE, sobald die konstante Geschwindigkeit erreicht ist und kann auch wieder FALSE werden, falls die Geschwindigkeit abweicht. Der Funktionsbaustein bleibt *Busy* und *Active* bis das Kommando abgelöst wird.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt solange TRUE wie der Baustein aktiv ist. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

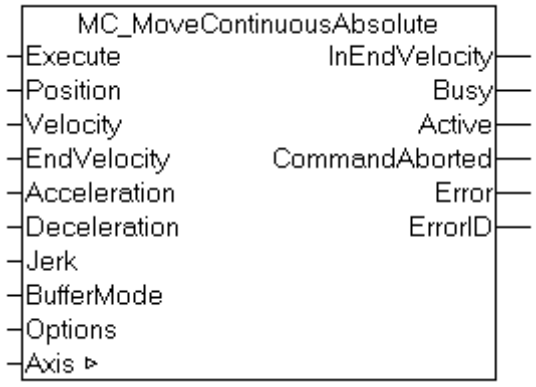
```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.1.7 MC_MoveContinuousAbsolute



MC_MoveContinuousAbsolute startet eine Positionierung auf eine absolute Zielposition und überwacht die Achsbewegung über den gesamten Fahrweg. An der Zielposition wird eine konstante Endgeschwindigkeit erreicht, die beibehalten wird. Der *InEndVelocity*-Ausgang wird gesetzt, wenn die Zielposition angefahren wurde. Anderenfalls wird der CommandAborted- oder im Fehlerfall der Error-Ausgang gesetzt.

Nachdem die Zielposition erreicht wurde, wird ist die Funktion des Bausteins abgeschlossen und die Achse wird nicht weiter überwacht.

Eingänge

```
VAR_INPUT
Execute : BOOL;
Position : LREAL;
Velocity : LREAL;
EndVelocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 107]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Position	Absolute Zielposition
Velocity	Maximale Geschwindigkeit mit der die Zielposition angefahren werden soll (>0).
EndVelocity	Endgeschwindigkeit, die nach Erreichen die Zielposition beibehalten werden soll.
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
InEndVelocity : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
```

```
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InEndVelocity	Der Ausgang InEndVelocity wird TRUE, wenn die Zielposition erreicht wurde. Der Funktionsbaustein bleibt *Busy* und *Active* bis das Kommando abgelöst wird.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

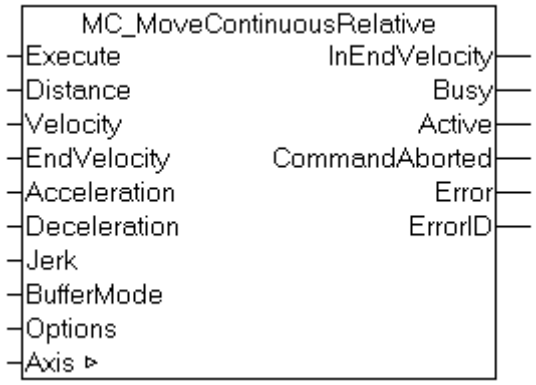
AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

MC_MoveContinuousAbsolute ist nicht implementiert für Eil-/Schleichachsen.

7.1.8 MC_MoveContinuousRelative



MC_MoveContinuousRelative startet eine Positionierung um eine relative Distanz und überwacht die Achsbewegung über den gesamten Fahrweg. An der Zielposition wird eine konstante Endgeschwindigkeit erreicht, die beibehalten wird. Der *InEndVelocity*-Ausgang wird gesetzt, wenn die Zielposition angefahren wurde. Anderenfalls wird der CommandAborted- oder im Fehlerfall der Error-Ausgang gesetzt.

Nachdem die Zielposition erreicht wurde, wird ist die Funktion des Bausteins abgeschlossen und die Achse wird nicht weiter überwacht.

Eingänge

```
VAR_INPUT
Execute : BOOL;
Distance : LREAL;
Velocity : LREAL;
EndVelocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 107]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Distance	Relative Fahrstrecke um die positioniert werden soll.
Velocity	Maximale Geschwindigkeit mit der *Distance* abgefahren werden soll (>0).
EndVelocity	Endgeschwindigkeit, die nach Abfahren der relativen Strecke *Distance* beibehalten werden soll
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom

	laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
InEndVelocity : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InEndVelocity	Der Ausgang InEndVelocity wird TRUE, wenn die Zielposition erreicht wurde. Der Funktionsbaustein bleibt *Busy* und *Active* bis das Kommando abgelöst wird.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann gestoppt worden sein oder das laufende Kommando wurde durch ein weiteres Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

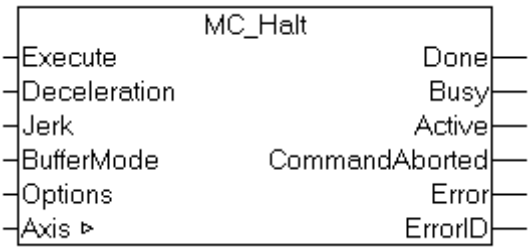
AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

MC_MoveContinuousRelative ist nicht implementiert für Eil-/Schleichachsen.

7.1.9 MC_Halt



MC_Halt hält eine Achse mit einer definierten Bremsrampe an.

Im Gegensatz zu MC_Stop [▶ 83] wird die Achse nicht gegen weitere Fahrbefehle verriegelt. Die Achse kann also sowohl während der Bremsrampe als auch nach dem Halt durch ein anderes Kommando gestartet werden.

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_Halt* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt. In diesem Fall ist ausschließlich der *Buffer-ModeAborting* möglich.

Eingänge

```
VAR_INPUT
Execute : BOOL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 107]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die mit dem letzten Move-Kommando parametrierte Verzögerung. *MC_Halt* und auch MC_Stop [▶ 83] können aus Sicherheitsgründen nicht mit schwächerer Dynamik ausgeführt werden, als der gerade aktive Fahrauftrag. Die Parametrierung wird gegebenenfalls automatisch angepasst.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der mit dem letzten Move-Kommando parametrierte Ruck. *MC_Halt* und auch MC_Stop [▶ 83] können aus Sicherheitsgründen nicht mit schwächerer Dynamik ausgeführt werden, als der gerade aktive Fahrauftrag. Die Parametrierung wird gegebenenfalls automatisch angepasst.
BufferMode	Der BufferMode [▶ 107] wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden

	Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den BufferMode festgelegt. Wird das Kommando auf eine gekoppelte Slave-Achse angewendet, so ist nur der Buffer-Mode *Aborting* möglich. Besonderheiten bei *MC_Halt*: Der Mode *MC_Buffered* zeigt keinen Effekt, das das Kommando erst im Stillstand ausgeführt wird. Die Blending Modes *MC_BlendingNext* und *MC_BlendingLow* verändern die letzte Zielposition nicht, können aber zu einer veränderten Dynamik (*Deceleration*) der Anhalterampe führen. Die Modes *MC_BlendingPrevious* und *MC_BlendingHigh* verlängern die Fahrt bis zur ursprünglichen Zielposition und leiten erst dort die Anhalterampe ein (definierter Bremspunkt).
Options	Zur Zeit nicht implementiert - Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Achse gestoppt wurde und im Stillstand ist.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange das Kommando abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Das laufende Kommando wurde eventuell durch ein Move-Kommando abgelöst.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

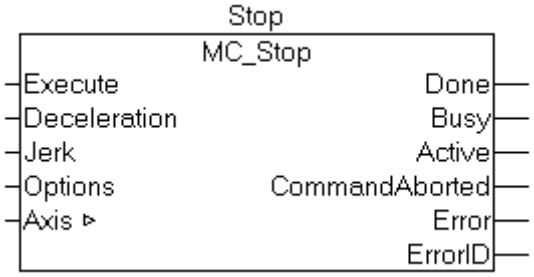
```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.1.10 MC_Stop



MC_Stop hält eine Achse mit einer definierten Bremsrampe an und verriegelt die Achse gegen andere Bewegungskommandos. Der Baustein eignet sich daher für Stopps in besonderen Situationen in denen eine weitere Bewegung der Achse unterbunden werden soll.

HINWEIS

Die Achse wird gleichzeitig für andere Bewegungskommandos gesperrt. Erst wenn nach dem vollständigen Stopp das Execute-Signal auf FALSE gesetzt wird, kann die Achse wieder gestartet werden. Die Entriegelung der Achse nach der fallenden Flanke von Execute benötigt wenige Zyklen. Während dieser Phase bleibt der Busy-Ausgang TRUE und der Funktionsbaustein muss weiter aufgerufen werden, bis Busy FALSE wird.

HINWEIS

Mit einem MC_Reset wird die Verriegelung der Achse aufgehoben.

Alternativ kann die Achse mit MC_Halt [▶ 81] ohne Verriegelung angehalten werden. *MC_Halt* ist für normale Bewegungsabläufe zu bevorzugen.

Bewegungskommandos können auf gekoppelte Slave-Achsen angewendet werden, wenn diese Option in den Parametern der Achse explizit aktiviert worden ist. Ein Bewegungskommando wie *MC_Stop* führt dann automatisch zum Abkoppeln der Achse und das Kommando wird anschließend ausgeführt.

Eingänge

```
VAR_INPUT
Execute : BOOL;
Deceleration : LREAL;
Jerk : LREAL;
Options : ST_MoveOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt. Die Achse wird während des Stopps verriegelt. Erst

	wenn nach dem vollständigen Stopp das Execute-Signal auf FALSE gesetzt wird, kann die Achse wieder gestartet werden.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die mit dem letzten Move-Kommando parametrierte Verzögerung. *MC_Stop* und auch *MC_Halt* können aus Sicherheitsgründen nicht mit schwächerer Dynamik ausgeführt werden, als der gerade aktive Fahrauftrag. Die Parametrierung wird gegebenenfalls automatisch angepasst.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der mit dem letzten Move-Kommando parametrierte Ruck. *MC_Stop* und auch *MC_Halt* können aus Sicherheitsgründen nicht mit schwächerer Dynamik ausgeführt werden, als der gerade aktive Fahrauftrag. Die Parametrierung wird gegebenenfalls automatisch angepasst.
Options	Zur Zeit nicht implementiert - Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Achse gestoppt wurde und im Stillstand ist.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange das Kommando abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. ***Hinweis*** **Busy bleibt TRUE solange, die Achse gesperrt ist. Erst nachdem Execute=FALSE gesetzt wird, wird die Achse entriegelt und Busy wird FALSE.**
Active	Active zeigt an, dass der Funktionsbaustein die Achse kontrolliert. ***Hinweis*** **Active bleibt TRUE solange, die Achse gesperrt ist. Erst nachdem Execute=FALSE gesetzt wird, wird die Achse entriegelt und Active wird FALSE.**
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

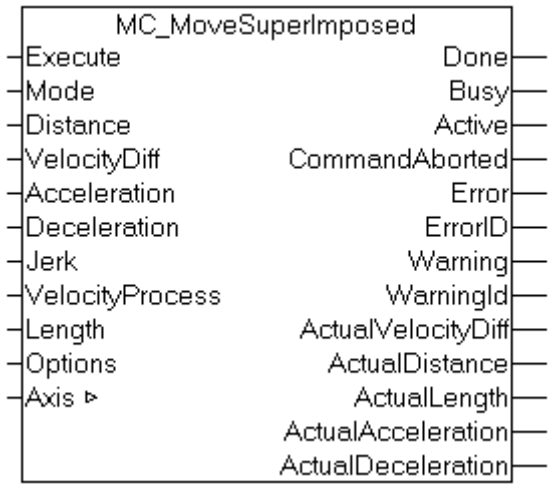
AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.2 Superposition

7.2.1 MC_MoveSuperimposed



MC_MoveSuperimposed startet eine relative überlagerte Bewegung während eine Achse bereits in Bewegung ist. Die aktuelle Bewegung wird nicht unterbrochen. Der *Done*-Ausgang wird gesetzt, wenn die überlagerte Bewegung beendet ist. Die ursprüngliche unterlagerte Bewegung kann weiterhin aktiv sein und wird durch den zugehörigen Move-Funktionsbaustein überwacht.

Die Funktion der Überlagerung wird deutlich, wenn man zwei gleich schnell laufende Achsen betrachtet. Wird eine der Achsen durch *MC_MoveSuperimposed* überlagert, so eilt sie um den Parameter *Distance* vor oder nach. Nachdem die überlagerte Bewegung beendet ist, bleibt die Streckendifferenz *Distance* zwischen beiden Achsen erhalten.

MC_MoveSuperimposed kann sowohl auf Einzelachsen, als auch auf Master- oder Slave-Achsen ausgeführt werden. Bei einer Slave-Achse wirkt die überlagerte Bewegung allein auf die Slave-Achse. Wird die Funktion auf eine Master-Achse angewendet, so macht der Slave aufgrund der Achskopplung die überlagerte Bewegung des Masters mit.

Da *MC_MoveSuperimposed* eine relative überlagerte Bewegung ausführt, verändert sich auch die Zielposition des unterlagerten Fahrkommandos um *Distance*.

Die überlagerte Bewegung wird abhängig von der Position der Hauptbewegung ausgeführt. Das heißt, dass bei einer Geschwindigkeitsänderung der Hauptbewegung auch die überlagerte Bewegung entsprechend langsamer oder schneller wird und bei einem Stillstand der Hauptbewegung ist auch die überlagerte Bewegung nicht mehr aktiv. Über den *Options*-Parameter kann festgelegt werden, ob die überlagerte Bewegung nach einem Stillstand der Hauptbewegung abgebrochen oder fortgesetzt wird.

Anwendungsbeispiele zu MC_MoveSuperimposed [▶ 88]

Eingänge

```
VAR_INPUT
Execute : BOOL; (* B *)
Mode : E_SuperpositionMode;
Distance : LREAL; (* B *)
VelocityDiff : LREAL; (* E *)
Acceleration : LREAL; (* E *)
Deceleration : LREAL; (* E *)
Jerk : LREAL; (* E *)
VelocityProcess : LREAL; (* V *)
Length : LREAL; (* V *)
Options : ST_SuperpositionOptions; (* V *)
END_VAR
```

ST_SuperpositionOptions [▶ 111] E_SuperpositionMode [▶ 110]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.	
Mode	Mode [▶ 110] legt fest, wie die überlagerte Bewegung auszuführen ist.	
Distance	Relative Wegstrecke die aufgeholt werden soll. Ein positiver Wert bedeutet eine betragsmäßige Geschwindigkeitserhöhung, um diese Strecke zusätzlich zur unbeeinflussten Bewegung zurückzulegen. Ein negativer Wert bedeutet ein Abbremsen und Zurückfallen um diese Strecke.	
VelocityDiff	Maximale Geschwindigkeitsdifferenz zur aktuellen Geschwindigkeit (Grundgeschwindigkeit) der Achse (>0). Bei diesem Parameter ist eventuell abhängig von der Überlagerungsrichtung (Beschleunigen oder Verzögern) eine Fallunterscheidung notwendig. Ist beispielsweise eine Fahrtrichtungsumkehr nicht erlaubt, so kann nur maximal bis zur Maximalgeschwindigkeit beschleunigt oder bis zum Stopp abgebremst werden. Demnach gibt es zwei Fälle für den maximal möglichen Wert von VelocityDiff: 1. Distance > 0 (Achse beschleunigt) VelocityDiff = Maximalgeschwindigkeit - Grundgeschwindigkeit 2. Distance < 0 (Achse verzögert) VelocityDiff = Grundgeschwindigkeit	
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.	
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.	
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.	
VelocityProcess	Mittlere Prozessgeschwindigkeit in der Achse (>0).Bei konstanter Grundgeschwindigkeit während der Überlagerung kann hier die Sollgeschwindigkeit der Achse angegeben werden.	
Length	Fahrstrecke, die für die überlagerte Bewegung zur Verfügung steht. Der Parameter *Mode* legt fest, wie diese Strecke interpretiert wird.	
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.	
Options.	**AbortOption**	AbortOption legt das Verhalten im Stillstand der unterlagerten Bewegung fest. Die überlagerte Bewegung kann dann abgebrochen oder später fortgesetzt werden.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
Warning : BOOL;
WarningID : UDINT;
ActualVelocityDiff : LREAL;
ActualDistance : LREAL;
ActualLength : LREAL;
ActualAcceleration : LREAL;
ActualDeceleration : LREAL;

END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die überlagerte Bewegung erfolgreich abgeschlossen wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird.
CommandAborted	Wird TRUE, wenn das Kommando durch ein anderes Kommando abgebrochen wurde und daher nicht vollständig ausgeführt werden konnte.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.
Warning	Warning wird TRUE, wenn die Aktion nicht vollständig ausgeführt werden kann.
WarningID	Der Baustein liefert die Warnung 4243_{hex} (16963) wenn die Ausgleichsfahrt aufgrund der Parametrierung (Strecke, Geschwindigkeit etc.) nicht vollständig durchgeführt werden kann. In diesem Fall wird die Ausgleichsfahrt soweit wie möglich ausgeführt. Der Anwender muss entscheiden, ob er diese Warnmeldung in seiner Applikation als echten Fehler oder eher als Warnung versteht.
ActualVelocityDiff	Tatsächlich genutzte Geschwindigkeitsüberhöhung während der überlagerten Bewegung (ActualVelocityDiff ≤ VelocityDiff).
ActualDistance	Tatsächlich überlagerte Distanz. Der Baustein versucht die vorgegebene Distanz *Distance* vollständig zu erreichen. Abhängig von der Parametrierung (*VelocityDiff, Acceleration, Deceleration, Length, Mode*) kann diese Distanz evtl. nicht vollständig erreicht werden. In diesen Fällen wird die maximal mögliche Distanz überlagert. (*ActualDistance≤Distance*).
ActualLength	Tatsächlich zurückgelegte Strecke während der überlagerten Bewegung (*ActualLenght≤Length*).
ActualAcceleration	Tatsächlich verwendete Beschleunigung der überlagerten Bewegung (*ActualAcceleration≤Acceleration*).

ActualDeceleration	Tatsächlich verwendete Verzögerung der überlagerten Bewegung (*ActualDeceleration≤Deceleration*).

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

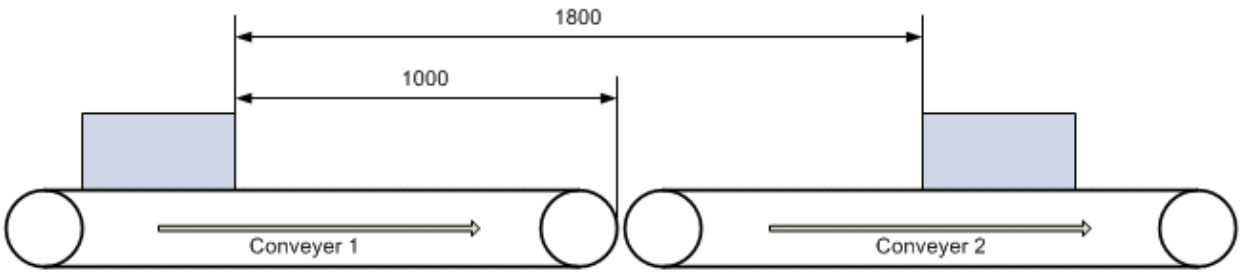
Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.2.2 Anwendungsbeispiele zu MC_MoveSuperimposed

Der Funktionsbaustein MC_MoveSuperimposed [▶ 85] startet einen überlagerte Bewegung auf einer bereits fahrenden Achse. Für diese Überlagerung gibt es verschiedene Anwendungsfälle, die im Folgenden beschrieben werden.

Abstandskorrektur für Produkte auf einem Förderband

Ein Förderband besteht aus einzelnen Segmenten, die jeweils durch eine Achse angetrieben werden. Auf dem Förderband werden Pakete transportiert, deren Abstand korrigiert werden soll. Dazu muss ein Fördersegment im Vergleich zu einem nachfolgenden Segment kurzzeitig schneller oder langsamer laufen.



Der gemessene Abstand ist 1800 mm und soll auf 1500 mm reduziert werden. Dazu soll Transportband Conveyer 1 beschleunigt werden, um den Abstand zu reduzieren. Die Korrektur muss bis zum Ende des Bandes 1 abgeschlossen sein, damit das Paket nicht auf das langsamer laufende Band 2 geschoben wird.

Da in dieser Situation Conveyer 1 beschleunigt werden muss, benötigt das Antriebssystem eine Geschwindigkeitsreserve, die hier mit 500 mm/s angenommen wird. In der Praxis lässt sich dieser Wert aus der Differenz zwischen maximaler Transportgeschwindigkeit und aktueller Sollgeschwindigkeit ermitteln.

Für die Parametrierung des Funktionsbausteins MC_MoveSuperimposed [▶ 85] bedeutet das:

Distance = 1800 mm - 1500 mm = 300 mm (Abstandskorrektur)

Length = 1000 mm (zur Verfügung stehende Strecke bis zum Ende des Bandes 1)

Mode = SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION

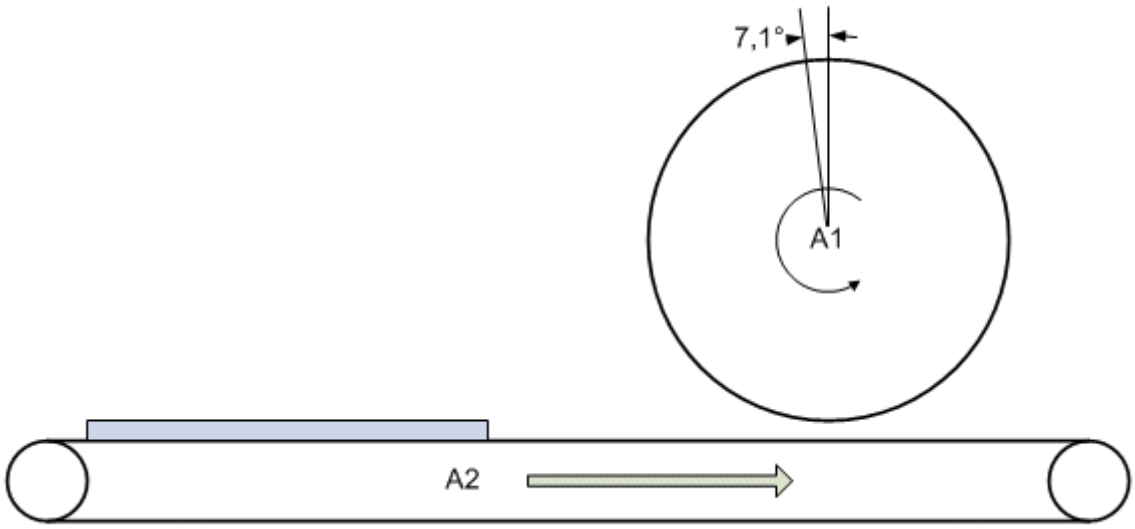
VelocityDiff = 500 mm/s

Der Mode legt fest, dass die Strecke *Length* bis zum Ende des Förderbandes zur Korrektur genutzt wird und dass die Korrektur an dieser Stelle abgeschlossen ist. Als Freiheitsgrad nutzt das System die Geschwindigkeit, die intern berechnet wird. *VelocityDiff* ist in diesem Fall also die Obergrenze für die Geschwindigkeitsänderung.

Alternativ könnte die Korrektur auch durch Abbremsen des Bandes 2 erreicht werden. In diesem Fall muss *Distance* negativ angegeben werden und die zur Verfügung stehende Korrekturstrecke *Length* ist der Abstand des rechten Paketes bis zum Bandende. Die maximal mögliche Geschwindigkeitsänderung *VelocityDiff* entspricht der aktuellen Sollgeschwindigkeit; das Band 2 könnte also, falls notwendig, bis zum Stillstand abgebremst werden.

Phasenverschiebung einer Druckwalze

Eine Druckwalze dreht mit konstanter Umfangsgeschwindigkeit gleich schnell wie ein Transportband auf dem ein zu bedruckendes Werkstück gefördert wird. Die Druckwalze soll zur Synchronisierung mit dem Werkstück um einen Winkel vorpositioniert werden (Phasenverschiebung).



Es gibt hier zwei Möglichkeiten, die Phasenverschiebung durchzuführen. Der Winkel kann so schnell wie möglich korrigiert werden wodurch die Geschwindigkeit der Druckwalze möglichst kurzzeitig und stark erhöht wird. Oder aber man definiert eine Korrekturstrecke innerhalb der die Korrektur stattfinden kann, z. B. eine volle Walzenumdrehung. Daraus ergeben sich folgende möglichen Parametrierungen des Funktionsbausteins MC_MoveSuperimposed [▶ 85]:

1. Schnelle Korrektur:

Distance = 7,1°

Length = 360° (maximal mögliche Korrekturstrecke)

Mode = SUPERPOSITIONMODE_LENGTHREDUCTION_LIMITEDMOTION

VelocityDiff = 30°/s (Geschwindigkeitsreserve)

Der *Mode* bestimmt, dass die Korrekturstrecke so weit wie möglich gekürzt wird. Der angegebene Wert für *Length* ist also eine Obergrenze, die frei aber nicht zu knapp gewählt werden kann.

Als *Mode* kann hier alternativ auch SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION verwendet werden. Die Gesamtstrecke für die Korrektur würde dann bei bis zu 367,1° liegen. Da die Strecke aber möglichst gekürzt wird, sind in diesem Fall beide Modi gleichwertig.

2. Langsame Korrektur:

Distance = 7,1°

Length = 360° (Korrekturstrecke)

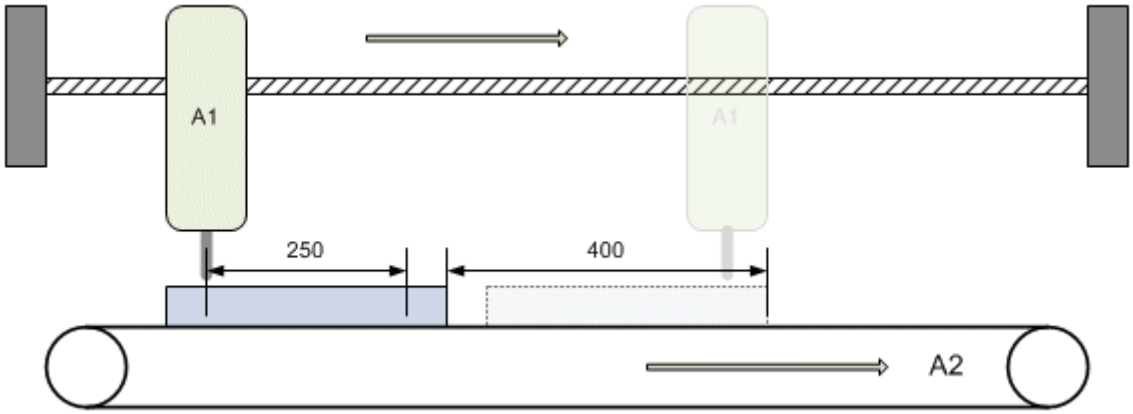
Mode = SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION

VelocityDiff = 30°/s (Geschwindigkeitsreserve)

Der *Mode* bestimmt, dass die Korrekturstrecke voll genutzt wird und die Geschwindigkeitsänderung möglichst klein gehalten wird. Der angegebene Wert für *VelocityDiff* ist also eine Obergrenze, die frei aber nicht zu knapp gewählt werden kann.

Bohraggregat

Ein Bohraggregat soll während des Transportes in ein Werkstück zwei Bohrungen setzen. Die Synchronisierung für die erste Bohrung soll hier nicht betrachtet werden und wurde z. B. mit der fliegenden Säge durchgeführt (MC_GearInPos). Nach der ersten Bearbeitung muss das Aggregat um eine bestimmte Distanz relativ zum fahrenden Werkstück versetzt werden.



Das Bohraggregat soll nach der ersten Bohrung um 250 mm relativ zum Werkstück vorpositioniert werden. Das Werkstück fährt in der Zeit eine Strecke von 400 mm ab. Ab dieser Position fährt das Bohraggregat wieder mit dem Werkstück synchron und die zweite Bearbeitung kann stattfinden.

Auch hier gibt es zwei mögliche Verfahren, die sich in der Geschwindigkeitsänderung des Bohraggregates und damit auch in der mechanischen Belastung unterscheiden.

Parametrierung des Funktionsbausteins MC_MoveSuperimposed [▶ 85]:

1. Schnelle Korrektur:

Distance = 250 mm

Length = 400 mm

Mode = SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION

VelocityDiff = 500 mm/s (Geschwindigkeitsreserve des Bohraggregats)

Der *Mode* bestimmt, dass die Strecke, auf der die Korrektur durchgeführt wird, so weit wie möglich gekürzt wird. Der angegebene Wert für *Length* ist also eine Obergrenze, die frei aber nicht zu knapp gewählt werden kann. Das Bohraggregat kann eine größere Strecke abfahren, da sich *Length* auf das Werkstück bezieht und relative Positionsänderung hinzukommt.

2. Langsame Korrektur:

Distance = 250 mm

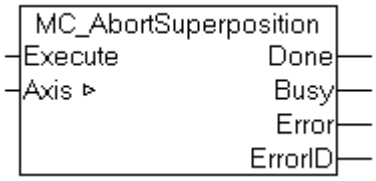
Length = 400 mm

Mode = SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION

VelocityDiff = 500 mm/s (Geschwindigkeitsreserve des Bohraggregats)

Der *Mode* bestimmt, dass die Korrekturstrecke voll genutzt wird und die Geschwindigkeitsänderung möglichst klein gehalten wird. Der angegebene Wert für *VelocityDiff* ist also eine Obergrenze, die frei aber nicht zu knapp gewählt werden kann. Das Werkstück fährt während der Positionsänderung des Bohraggregates die Strecke *Length* ab, das Aggregat positioniert wegen der zusätzlichen Korrekturstrecke um 650 mm (*Length* + *Distance*).

7.2.3 MC_AbortSuperposition



Der Baustein *MC_AbortSuperposition* bricht eine durch MC_MoveSuperImposed [▶ 85] gestartete überlagerte Bewegung ab, ohne die unterlagerte Achsbewegung zu stoppen.

Ein vollständiger Achsstopp kann gegebenenfalls mit MC_Stop [▶ 83] oder MC_Halt [▶ 81] durchgeführt werden. Ein Aufruf von *MC_AbortSuperposition* ist dann nicht notwendig.

Eingänge

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	Mit der steigenden Flanke wird das Kommando ausgeführt und die überlagerte Bewegung beendet.

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Wird TRUE, sobald die überlagerte Bewegung erfolgreich abgebrochen wurde.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet.
Error	Wird TRUE, sobald ein Fehler auftritt.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

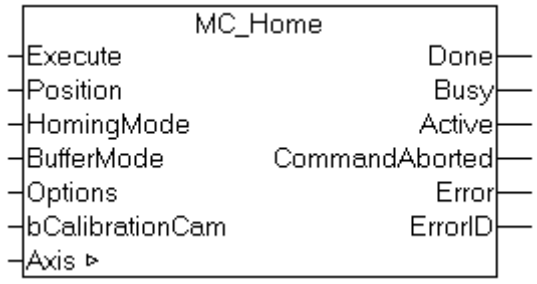
AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.3 Homing

7.3.1 MC_Home



Mit dem Funktionsbaustein *MC_Home* wird eine Referenzierfahrt der Achse durchgeführt.

Der Referenziermodus wird im TwinCAT SystemManager mit dem Encoderparameter *Reference Mode* eingestellt. Abhängig vom angeschlossenen Encoder-System sind verschiedene Abläufe möglich (siehe auch Referenziermodus für Inkrementalencoder)

Eingänge

```
VAR_INPUT
Execute : BOOL;
Position : LREAL := DEFAULT_HOME_POSITION;
HomingMode : MC_HomingMode;
BufferMode : MC_BufferMode;
Options : ST_HomingOptions;
bCalibrationCam : BOOL;
END_VAR
```

MC_BufferMode [▶ 107] MC_HomingMode [▶ 110]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Position	Absolute Referenzposition auf die die Achse nach der Referenzfahrt gesetzt wird. Alternativ kann hier die Konstante DEFAULT_HOME_POSITION verwendet werden. Dadurch wird die im TwinCAT System Manager festgelegte *Referenzposition für Referenzierfahrt* verwendet. ***Hinweis*** **Da die Referenzposition üblicherweise noch während der Fahrt gesetzt wird, bleibt die Achse nicht exakt an dieser Position stehen. Die Stillstandsposition weicht um den Bremsweg der Achse ab, dennoch ist die Kalibrierung exakt.**
HomingMode	HomingMode [▶ 110] bestimmt, auf welche Weise die Kalibrierung durchgeführt wird. • MC_DefaultHoming Führt die Standard-Referenzfahrt aus. • MC_Direct Setzt die Position der Achse direkt auf *Position* ohne eine Bewegung auszuführen. • MC_ForceCalibration Erzwingt den Zustand "Achse ist kalibriert". Es wird keine Bewegung ausgeführt und die Position bleibt unverändert. • MC_ResetCalibration Setzt den Kalibrierungszustand der Achse zurück. Es wird keine Bewegung ausgeführt und die Position bleibt unverändert.

BufferMode	Zur Zeit nicht implementiert - Der *BufferMode* wird ausgewertet, wenn die Achse bereits ein anderes Kommando ausführt. Das laufende Kommando kann abgebrochen werden oder dieses Kommando wird erst nach dem laufenden Kommando aktiv. Die Übergangsbedingung vom laufenden zum nächsten Kommando wird ebenfalls durch den *BufferMode* festgelegt.	
Options	Die Datenstruktur Options enthält zusätzliche, selten benötigte Parameter. Im Normalfall kann der Eingang offen bleiben.	
Options.	**ClearPositionLag**	ClearPositionLag wirkt nur im Mode MC_Direct. *ClearPositionLag* kann optional gesetzt werden, falls Soll- und Istposition auf den gleichen Wert gesetzt werden sollen. Damit wird der Schleppfehler gelöscht.
bCalibrationCam	bCalibrationCam spielgelt das Signal einer Referenznocke wieder, das über einen digitalen Eingang in die Steuerung kommen kann.	

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Der Ausgang *Done* wird TRUE, wenn die Achse kalibriert wurde und die Bewegung beendet ist.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Fahrbefehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done*, *CommandAborted* oder *Error* gesetzt.
Active	Zur Zeit nicht implementiert - Active zeigt an, dass das Kommando ausgeführt wird. Wenn das Kommando gepuffert wurde, wird es evtl. erst aktiv, nachdem ein laufendes Kommando beendet ist.
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105]

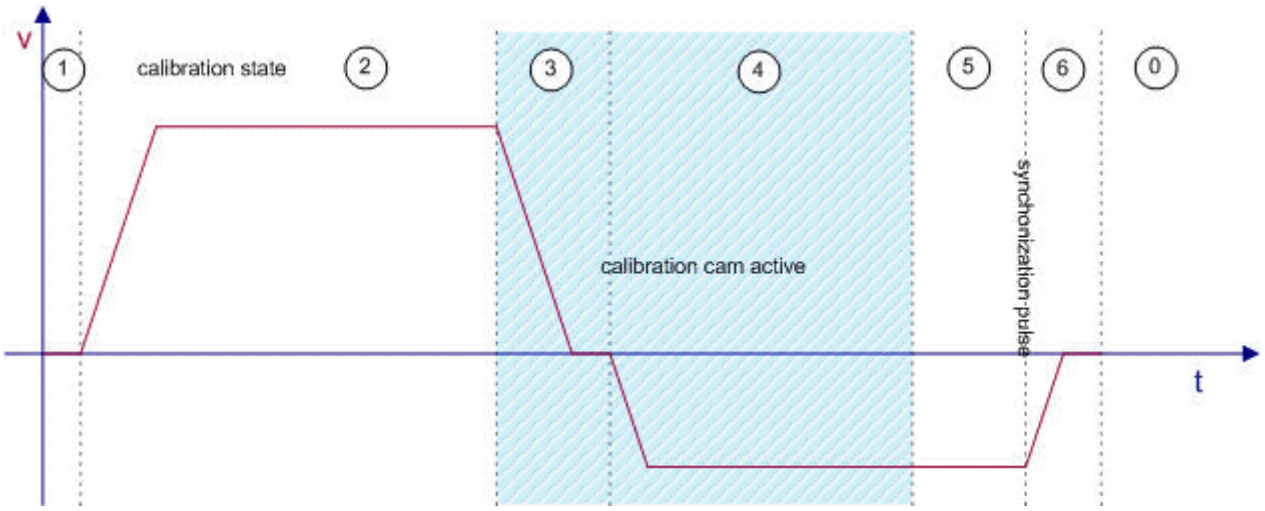
Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

Anmerkung

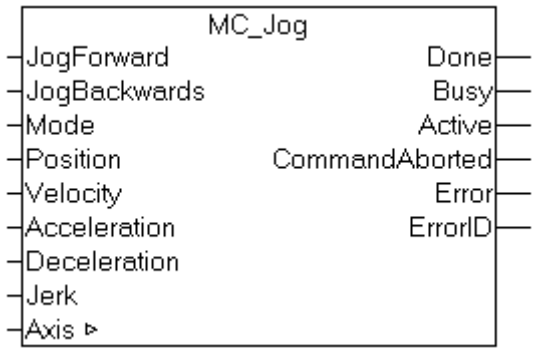
Der Referenziervorgang läuft in mehreren Phasen ab. Der Referenzierstatus (Calibration State) wird im zyklischen Interface der Achse signalisiert (Axis.NcToPlc.HomingState). Im nachfolgenden Bild ist der Ablauf nach dem Start des *MC_Home* Bausteins mit den einzelnen Phasen schematisch dargestellt.

Soll eine Achse ohne Referenznocke, also nur auf den Sync-Impuls des Gebers, referenziert werden, so kann die Referenznocke durch das SPS-Programm simuliert werden. Das Signal bCalibrationCam wird zunächst aktiviert und dann zurückgenommen, wenn Axis.NcToPlc.HomingState [▶ 106] größer oder gleich 4 ist.



7.4 Manual Motion

7.4.1 MC_Jog



Der Funktionsbaustein *MC_Jog* ermöglicht es, eine Achse mit Handbedientasten zu fahren. Das Tastensignal kann direkt mit den beiden Eingängen *JogForward* und *JogBackwards* verbunden werden. Über den Mode-Eingang wird die gewünschte Betriebsart festgelegt. So steht beispielsweise auch ein Inching-Mode zur Verfügung, in dem die Achse mit jedem Tastendruck um einen festgelegten Distanz-Schritt fährt. Je nach Betriebsart können Geschwindigkeit und Dynamik der Bewegung bestimmt werden.

Eingänge

```
VAR_INPUT
JogForward : BOOL;
JogBackwards : BOOL;
Mode : E_JogMode;
Position : LREAL;
```

```
Velocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
END_VAR
```

E_JogMode [▶ 112]

JogForward	Mit der steigenden Flanke wird das Kommando ausgeführt und die Achse wird in positiver Fahrtrichtung bewegt.Je nach Betriebsart (siehe Mode), fährt die Achse solange das Signal TRUE bleibt oder stoppt automatisch nach einer festgelegten Distanz. Während der Bewegung werden keine weiteren Signalflanken angenommen, auch nicht am Eingang JogBackwards. Bei gleichzeitiger Signalflanke an den Eingängen JogForward und JogBackwards hat JogForward Vorrang.
JogBackwards	Mit der steigenden Flanke wird das Kommando ausgeführt und die Achse wird in negativer Fahrtrichtung bewegt. JogForward und JogBackwards sollten alternativ getriggert werden, sind aber auch intern gegeneinander verriegelt.
Mode	Der Mode [▶ 112]-Eingang legt die Betriebsart fest, in der die Handfunktion ausgeführt wird. *MC_JOGMODE_STANDARD_SLOW:* Die Achse wird solange verfahren, wie das Signal an einem der Jog-Eingänge TRUE ist. Dabei wird die im TwinCAT SystemManager festgelegte *niedrige Geschwindigkeit für Handfunktionen* und die Standarddynamik verwendet. In dieser Betriebsart haben die am Funktionsbaustein angelegten Positions-, Geschwindigkeits- und Dynamikdaten keine Bedeutung. *MC_JOGMODE_STANDARD_FAST:* Die Achse wird solange verfahren, wie das Signal an einem der Jog-Eingänge TRUE ist. Dabei wird die im TwinCAT SystemManager festgelegte *hohe Geschwindigkeit für Handfunktionen* und die Standarddynamik verwendet. In dieser Betriebsart haben die am Funktionsbaustein angelegten Positions-, Geschwindigkeits- und Dynamikdaten keine Bedeutung. *MC_JOGMODE_CONTINOUS:* Die Achse wird solange verfahren, wie das Signal an einem der Jog-Eingänge TRUE ist. Dabei werden die vom Anwender angegebenen Geschwindigkeits- und Dynamikdaten verwendet. Die Position hat keine Bedeutung. *MC_JOGMODE_INCHING:* Die Achse wird mit steigender Flanke an einem der Jog-Eingänge um eine bestimmte Distanz verfahren, die über den Positions-Eingang festgelegt wird. Die Achse stoppt automatisch, unabhängig vom Zustand der Jog-Eingänge. Erst mit einer weiteren steigenden Flanke wird ein neuer Bewegungsschritt ausgeführt. Mit jedem Start werden die vom Anwender angegebenen Geschwindigkeits- und Dynamikdaten verwendet. *MC_JOGMODE_INCHING_MODULO:* Die Achse wird mit steigender Flanke an einem der Jog-Eingänge um eine bestimmte Distanz verfahren, die über den Positions-Eingang festgelegt wird. Die

	Achsposition rastet dabei auf ein ganzzahliges Vielfaches des Positionsparameters ein. Die Achse stoppt automatisch, unabhängig vom Zustand der Jog-Eingänge. Erst mit einer weiteren steigenden Flanke wird ein neuer Bewegungsschritt ausgeführt. Mit jedem Start werden die vom Anwender angegebenen Geschwindigkeits- und Dynamikdaten verwendet.
Position	relative Distanz, um die in der Betriebsart *MC_JOGMODE_INCHING* verfahren wird.
Velocity	Maximale Geschwindigkeit mit der gefahren werden soll (>0).
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager.
Deceleration	Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager.
Jerk	Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager.

Die Parameter *Position*, *Velocity*, *Acceleration*, *Deceleration* und *Jerk* werden in den Betriebsarten *MC_JOGMODE_STANDARD_SLOW* und *MC_JOGMODE_STANDARD_FAST* nicht verwendet und können frei bleiben.

Ausgänge

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Wird TRUE wenn eine Bewegung erfolgreich abgeschlossen wurde.
Busy	Wird TRUE sobald der Baustein aktiv ist und wird FALSE nachdem er sich wieder im Grundzustand befindet. Erst dann kann eine weitere Flanke an den Jog-Eingängen angenommen werden.
Active	Active zeigt an, dass die Achse durch die Jog-Funktion bewegt wird.
CommandAborted	Wird TRUE wenn der Vorgang von außen, z. B. durch den Aufruf von MC_Stop [▶ 83], abgebrochen wurde.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

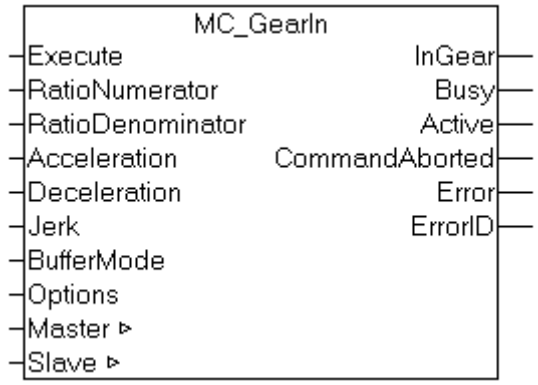
AXIS_REF [▶ 105]

Axis	Achsdatenstruktur

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.5 Achskopplung

7.5.1 MC_GearIn



Mit dem Funktionsbaustein *MC_GearIn* wird ein lineare Master-Slave-Kopplung (Getriebekopplung) aktiviert. Der Baustein akzeptiert einen festen Getriebefaktor im Zähler-Nenner-Format.

Die Slave-Achse kann im Stillstand an die Master-Achse gekoppelt werden. Es ist mit diesem Baustein nicht möglich, auf eine fahrende Master-Achse aufzusynchronisieren. Zu diesem Zweck kann der *Fliegende Säge* Baustein MC_GearInVelo oder MC_GearInPos verwendet werden.

Die Slave-Achse kann mit dem Funktionsbaustein MC_GearOut [▶ 100] abgekoppelt werden. Wird der Slave während der Fahrt abgekoppelt, so behält er seine Geschwindigkeit bei und kann mit MC_Stop [▶ 83] oder MC_Halt [▶ 81] angehalten werden.

Alternativ steht der Baustein MC_GearInDyn [▶ 99] mit dynamisch änderbarem Getriebefaktor zur Verfügung.

Eingänge

```
VAR_INPUT
Execute : BOOL;
RatioNumerator : LREAL;
RatioDenominator : UINT;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_GearInOptions;
END_VAR
```

MC_BufferMode [▶ 107]

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
RatioNumerator	Getriebefaktor Zähler. Alternativ kann der Getriebefaktor im Zähler als Fließkommawert angegeben werden, wenn der Nenner 1 ist.
RatioDenominator	Getriebefaktor Nenner
Acceleration	Beschleunigung (≥0). (zur Zeit nicht implementiert)
Deceleration	Verzögerung (≥0). (zur Zeit nicht implementiert)
Jerk	Ruck (≥0). (zur Zeit nicht implementiert)

BufferMode	Zur Zeit nicht implementiert
Options	Zur Zeit nicht implementiert

Bei einem Verhältnis 1:4 muss der *RatioNumerator* 1 sein und der *RatioDenominator* 4. Alternativ kann der *RatioDenominator* 1 sein und der Getriebefaktor wird im *RatioNumerator* als Fließkommazahl 0.25 angegeben. Der *RatioNumerator* darf negativ sein.

Ausgänge

```
VAR_OUTPUT
InGear : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InGear	Wird TRUE, wenn die Kopplung erfolgreich durchgeführt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *InGear*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. (zur Zeit ist Active=Busy, siehe BufferMode)
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann während des Koppelvorgangs entkoppelt worden sein (gleichzeitige Kommandoausführung).
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

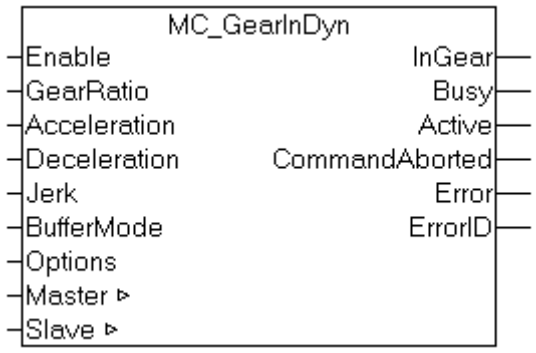
```
VAR_IN_OUT
Master : AXIS_REF;
Slave : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105] AXIS_REF [▶ 105]

Master	Achsdatenstruktur des Masters.
Slave	Achsdatenstruktur des Slaves.

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.5.2 MC_GearInDyn



Mit dem Funktionsbaustein *MC_GearInDyn* wird ein lineare Master-Slave-Kopplung (Getriebekopplung) aktiviert. Der Getriebefaktor kann dynamisch, d. h. in jedem SPS-Zyklus angepasst werden. Somit lässt sich eine geregelte Master-Slave-Kopplung aufbauen. Der Parameter *Acceleration* wirkt begrenzend, falls die Änderungen des Getriebefaktors sehr groß sind.

Die Slave-Achse kann mit dem Funktionsbaustein MC_GearOut [▶ 100] abgekoppelt werden. Wird der Slave während der Fahrt abgekoppelt, so behält er seine Geschwindigkeit bei und kann mit MC_Stop [▶ 83] oder MC_Halt [▶ 81] angehalten werden.

Alternativ steht der Baustein MC_GearIn [▶ 97] mit festem Getriebefaktor zur Verfügung.

Eingänge

```
VAR_INPUT
Enable : BOOL;
GearRatio : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_GearInDynOptions;
END_VAR
```

MC_BufferMode [▶ 107]

Enable	Mit einer steigenden Flanke am Eingang *Enable* führt die Kopplung aus. Solange *Enable* TRUE ist, kann der Getriebefaktor zyklisch geändert werden. Wenn *Enable* nach der Kopplung FALSE wird, wird das Kommando beendet. Der Getriebefaktor wird auf seinem letzten Wert eingefroren aber der Slave wird nicht entkoppelt.
GearRatio	Getriebefaktor als Fließkommawert. Der Getriebefaktor kann zyklisch geändert werden, solange *Enable* TRUE ist. Wenn *Enable* FALSE ist, bleibt der Getriebefaktor unverändert.
Acceleration	Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager. Der Parameter begrenzt die Beschleunigung des Slaves bei großen Änderungen des Getriebefaktors. Die maximale Beschleunigung wird erst bei maximaler Master-Geschwindigkeit erreicht, anderenfalls liegt die Slave-Beschleunigung bei großen Getriebefaktoränderungen unterhalb dieses Wertes.
Deceleration	Verzögerung (≥0). (nicht implementiert)

Jerk	Ruck (≥0). (nicht implementiert)
BufferMode	Zur Zeit nicht implementiert
Options	Zur Zeit nicht implementiert

Ausgänge

```
VAR_OUTPUT
InGear : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InGear	Wird TRUE, wenn die Kopplung erfolgreich durchgeführt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Enable* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *InGear*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. (zur Zeit ist Active=Busy, siehe BufferMode)
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann während des Koppelvorgangs entkoppelt worden sein (gleichzeitige Kommandoausführung).
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

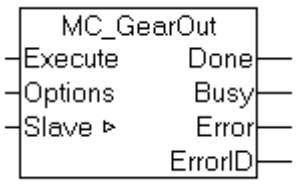
```
VAR_IN_OUT
Master : AXIS_REF;
Slave : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105] AXIS_REF [▶ 105]

Master	Achsdatenstruktur des Masters.
Slave	Achsdatenstruktur des Slaves.

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.5.3 MC_GearOut



Mit dem Funktionsbaustein *MC_GearOut* wird ein Master-Slave-Kopplung deaktiviert.

⚠ WARNUNG

Kein Stillstand der Achse durch Abkopplung

Wenn eine Slaveachse in der Bewegung abgekoppelt wird, wird sie nicht automatisch gestoppt, sondern erreicht eine konstante Geschwindigkeit mit der sie endlos weiterfährt.

Sie können die Achse mit den Bausteinen MC_Halt [▶ 81] oder MC_Stop [▶ 83] anhalten.

HINWEIS

Wenn der Sollwertgeneratortype der Achse auf *"7 Phasen (optimiert)"* eingestellt ist, wird die Slaveachse nach dem Abkoppeln beschleunigungsfrei gefahren und mit der sich einstellenden konstanten Geschwindigkeit weitergefahren. Es erfolgt keine Positionierung um den mit dem Koppelfaktor umgerechneten Masterverfahrweg, sondern es stellt sich ein Verhalten wie nach einem MC_MoveVelocity ein. In TwinCAT 2.10 ist der Sollwertgeneratortyp wählbar. Ab TwinCAT 2.11 ist der Sollwertgeneratortype fest auf *"7 Phasen (optimiert)"* eingestellt. Bei der Umstellung eines Projektes von TwinCAT 2.10 auf TwinCAT 2.11 ergibt sich damit das hier beschriebene Verhalten. Ein Update bestehender Applikationen auf Version 2.11 kann daher, je nach Anwendung, eine Anpassung des SPS-Programms erforderlich machen.

Eingänge

```
VAR_INPUT
    Execute  : BOOL;
    Options  : ST_GearOutOptions;
END_VAR
```

Execute	Mit einer steigenden Flanke am Eingang *Execute* wird das Kommando ausgeführt.
Options	Zurzeit nicht implementiert

Ausgänge

```
VAR_OUTPUT
    Done     : BOOL;
    Busy     : BOOL;
    Error    : BOOL;
    ErrorID  : UDINT;
END_VAR
```

Done	Wird TRUE, wenn die Achse erfolgreich abgekoppelt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Execute* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn *Busy* wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *Done* oder *Error* gesetzt.
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

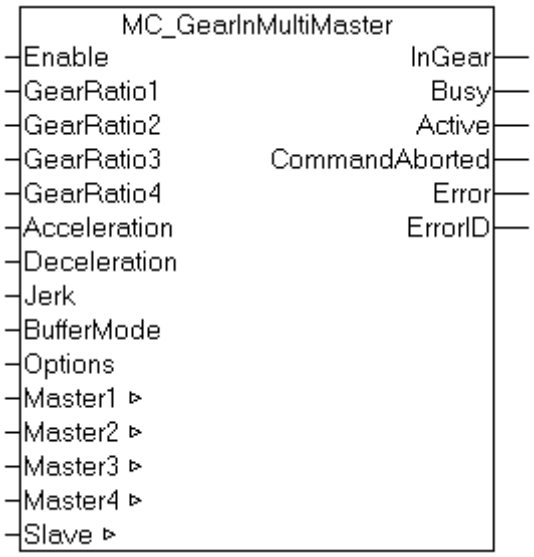
Ein/Ausgänge

```
VAR_IN_OUT
    Slave : AXIS_REF;
END_VAR
```

Slave	Achsdatenstruktur des Slaves [▶ 105].

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

7.5.4 MC_GearInMultiMaster



Mit dem Funktionsbaustein *MC_GearInMultiMaster* wird ein lineare Master-Slave-Kopplung (Getriebekopplung) zu bis zu vier verschiedenen Master-Achsen aktiviert. Der Getriebefaktor kann jeweils dynamisch, d. h. in jedem SPS-Zyklus angepasst werden. Die Slave-Bewegung ergibt sich aus der Überlagerung der Bewegungen der Master. Der Parameter *Acceleration* wirkt begrenzend, falls die Änderungen des Getriebefaktors sehr groß sind.

Die Slave-Achse kann mit dem Funktionsbaustein MC_GearOut [▶ 100] abgekoppelt werden. Wird der Slave während der Fahrt abgekoppelt, so behält er seine Geschwindigkeit bei und kann mit MC_Stop [▶ 83] angehalten werden.

Falls weniger als vier Master verwendet werden, kann für die Parameter *Master2* bis *Master4* jeweils eine leere Datenstruktur übergeben werden (die Achs-ID muss 0 sein).

Eingänge

```
VAR_INPUT
Enable : BOOL;
GearRatio1 : LREAL;
GearRatio2 : LREAL;
GearRatio3 : LREAL;
GearRatio4 : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_GearInMultiMasterOptions;
END_VAR
```

Enable	Mit einer steigenden Flanke am Eingang *Enable* führt die Kopplung aus. Solange *Enable* TRUE ist, kann der Getriebefaktor zyklisch geändert werden. Wenn *Enable* nach der Kopplung FALSE wird, wird das Kommando beendet. Der Getriebefaktor wird auf seinem letzten Wert eingefroren aber der Slave wird nicht entkoppelt.
GearRatio1	Getriebefaktor als Fließkommawert für die erste Master-Achse. Der Getriebefaktor kann zyklisch geändert werden, solange *Enable* TRUE ist. Wenn *Enable* FALSE ist, bleibt der Getriebefaktor unverändert.

GearRatio2		Getriebefaktor als Fließkommawert für die zweite Master-Achse. Der Getriebefaktor kann zyklisch geändert werden, solange *Enable* TRUE ist. Wenn *Enable* FALSE ist, bleibt der Getriebefaktor unverändert.
GearRatio3		Getriebefaktor als Fließkommawert für die dritte Master-Achse. Der Getriebefaktor kann zyklisch geändert werden, solange *Enable* TRUE ist. Wenn *Enable* FALSE ist, bleibt der Getriebefaktor unverändert.
GearRatio4		Getriebefaktor als Fließkommawert für die vierte Master-Achse. Der Getriebefaktor kann zyklisch geändert werden, solange *Enable* TRUE ist. Wenn *Enable* FALSE ist, bleibt der Getriebefaktor unverändert.
Acceleration		Beschleunigung (≥0). Bei einem Wert von 0 wirkt die Standardbeschleunigung aus der Achskonfiguration im System Manager. Der Parameter begrenzt die Beschleunigung des Slaves bei großen Änderungen des Getriebefaktors.
Deceleration		Verzögerung (≥0). Bei einem Wert von 0 wirkt die Standardverzögerung aus der Achskonfiguration im System Manager. Der Parameter begrenzt die Verzögerung des Slaves bei großen Änderungen des Getriebefaktors. **Nur für die Option „AdvancedSlaveDynamics“ verwendet.**
Jerk		Ruck (≥0). Bei einem Wert von 0 wirkt der Standardruck aus der Achskonfiguration im System Manager. Der Parameter begrenzt den Ruck des Slaves bei großen Änderungen des Getriebefaktors. **Nur für die Option „AdvancedSlaveDynamics“ verwendet.**
BufferMode		Zur Zeit nicht implementiert
Options.	**AdvancedSlaveDynamics**	Tauscht den internen Algorithmus des Bausteins. Damit ist es möglich auf bereits in Bewegung befindliche Master aufzusynchronisieren. Dabei sollten Acceleration und Deceleration ausschließlich symmetrisch parametriert werden. Bei zu großen Ruck-Vorgaben wird dieser soweit reduziert, dass eine Änderung von Null auf die parametrierte Beschleunigung / Verzögerung in einem NC-Zyklus erfolgen kann. Die Auflösung der Beschleunigung / Verzögerung hängt somit direkt von der geeigneten Parametrierung des Ruck-Wertes ab.

Ausgänge

```
VAR_OUTPUT
InGear : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InGear	Wird TRUE, wenn die Kopplung erfolgreich durchgeführt wurde.
Busy	Der *Busy*-Ausgang wird TRUE, sobald das Kommando mit *Enable* gestartet wird und bleibt TRUE, solange der Befehl abgearbeitet wird. Wenn

	Busy wieder FALSE wird, so ist der Funktionsbaustein bereit für einen neuen Auftrag. Gleichzeitig ist einer der Ausgänge *InGear*, *CommandAborted* oder *Error* gesetzt.
Active	Active zeigt an, dass das Kommando ausgeführt wird. (zur Zeit ist Active=Busy, siehe BufferMode)
CommandAborted	Wird TRUE, wenn das Kommando nicht vollständig ausgeführt werden konnte. Die Achse kann während des Koppelvorgangs entkoppelt worden sein (gleichzeitige Kommandoausführung).
Error	Wird im Fehlerfall TRUE.
ErrorID	Liefert bei einem gesetzten Error-Ausgang die Fehlernummer.

Ein/Ausgänge

```
VAR_IN_OUT
Master1 : AXIS_REF;
Master2 : AXIS_REF;
Master3 : AXIS_REF;
Master4 : AXIS_REF;
Slave : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 105] AXIS_REF [▶ 105] AXIS_REF [▶ 105] AXIS_REF [▶ 105] AXIS_REF [▶ 105]

Master1	Achsdatenstruktur des ersten Masters.
Master2	Achsdatenstruktur des zweiten Masters.
Master3	Achsdatenstruktur des dritten Masters.
Master4	Achsdatenstruktur des vierten Masters.
Slave	Achsdatenstruktur des Slaves.

Die Achsdatenstruktur vom Typ AXIS_REF [▶ 105] adressiert eine Achse eindeutig im System. Sie enthält unter anderem den aktuellen Status der Achse, wie Position, Geschwindigkeit oder Fehlerzustand.

8 Datentypen

8.1 Achsinterface

8.1.1 Datentyp AXIS_REF

Der Datentyp AXIS_REF enthält Information zu einer Achse. AXIS_REF ist eine Schnittstelle zwischen der SPS und der NC und wird den MC-Funktionsbausteinen als Referenz auf eine Achse mitgegeben.

```
TYPE AXIS_REF :
VAR_INPUT
PlcToNc    AT %Q* : PLCTONC_AXIS_REF;
END_VAR
VAR_OUTPUT
NcToPlc    AT %I* : NCTOPLC_AXIS_REF;
ADS               : ST_AdsAddress;
Status            : ST_AxisStatus;
END_VAR
END_TYPE
```

ST_AxisStatus [▶ 113] NCTOPLC_AXIS_REF [▶ 106] PLCTONC_AXIS_REF [▶ 106]

Elemente von AXIS_REF

PlcToNc : PlcToNc [▶ 106] ist eine Datenstruktur, die zyklisch zwischen SPS und NC ausgetauscht wird. Über diese Datenstruktur kommunizieren die MC-Funktionsbausteine mit der NC und senden Kontrollinformation von der SPS zur NC. Diese Datenstruktur wird automatisch im Ausgangsprozessabbild der SPS platziert und muss im TwinCAT System Manager mit dem Eingangsprozessabbild einer NC-Achse verbunden werden.

NcToPlc : NcToPlc [▶ 106] ist eine Datenstruktur, die zyklisch zwischen SPS und NC ausgetauscht wird. Über diese Datenstruktur kommunizieren die MC-Funktionsbausteine mit der NC und empfangen Statusinformationen von der NC. Diese Datenstruktur wird automatisch im Eingangsprozessabbild der SPS platziert und muss im TwinCAT System Manager mit dem Ausgangsprozessabbild einer NC-Achse verbunden werden.

Die NcToPlc [▶ 106] -Struktur enthält alle wesentlichen Zustandsinformationen einer Achse wie Position, Geschwindigkeit und Auftragszustand. Da der Datenaustausch zyklisch stattfindet, kann die SPS jederzeit ohne zusätzlichen Kommunikationsaufwand auf den aktuellen Achszustand zugreifen.

ADS : die ADS-Datenstruktur enthält die ADS-Kommunikationsparameter einer Achse, die für eine direkte ADS-Kommunikation benötigt werden. Im Normalfall muss diese Struktur nicht belegt werden. Erst wenn eine Achse auf einem anderen Zielsystem, oder über eine besondere Port-Nummer angesprochen werden soll, kann der Anwender hier die entsprechende Information hinterlegen.

Status : Die Status-Datenstruktur [▶ 113] enthält zusätzliche oder aufbereitete Statusinformation zu einer Achse. Diese Datenstruktur wird nicht zyklisch aufgefrischt, sondern muss durch das SPS-Programm aktualisiert werden. Am einfachsten geschieht das durch einen Aufruf von MC_ReadStatus [▶ 32] oder alternativ durch den Aufruf der Aktion *ReadStatus* von AXIS_REF:

Beispiel:

```
VAR
Axis1 : AXIS_REF (* axis data structure for Axis-1 *)
END_VAR

(* program code at the beginning of each PLC cycle *)
Axis1.ReadStatus;
```

Der Aufruf von *ReadStatus* sollte einmalig am Anfang jedes SPS-Zyklus getätigt werden. Anschließend kann innerhalb des gesamten SPS-Programms auf die aktuelle Statusinformation in AXIS_REF zugegriffen werden. Innerhalb eines Zyklus ändert sich der Status nicht.

8.1.2 Datentyp NCTOPLC_AXIS_REF

Die Datenstruktur NCTOPLC_AXIS_REF ist Bestandteil der AXIS_REF [▶ 105] Datenstruktur und wird automatisch von der NC aktualisiert, so dass die Informationen in jedem SPS-Zyklus aktuell vorliegen. NCTOPLC_AXIS_REF wir auch als Achsinterface zwischen NC und SPS bezeichnet.

```
TYPE NCTOPLC_AXIS_REF
STRUCT
StateDWord            : DWORD; (* Status double word *)
ErrorCode             : DWORD; (* Axis error code *)
AxisState             : DWORD; (* Axis moving status *)
AxisModeConfirmation  : DWORD; (* Axis mode confirmation (feedback from NC) *)
HomingState           : DWORD; (* State of axis calibration (homing) *)
CoupleState           : DWORD; (* Axis coupling state *)
SvbEntries            : DWORD; (* SVB entries/orders (SVB = Set preparation task) *)
SafEntries            : DWORD; (* SAF entries/orders (SAF = Set execution task) *)
AxisId                : DWORD; (* Axis ID *)
OpModeDWord           : DWORD; (* Current operation mode *)
ActiveControlLoopIndex: WORD;  (* Active control loop index *)
ControlLoopIndex      : WORD;  (* Axis control loop index (0, 1, 2, when multiple control loops are
used) *)
ActPos                : LREAL; (* Actual position (absolut value from NC) *)
ModuloActPos          : LREAL; (* Actual modulo position *)
ModuloActTurns        : DINT;  (* Actual modulo turns *)
ActVelo               : LREAL; (* Actual velocity *)
PosDiff               : LREAL; (* Position difference (lag distance) *)
SetPos                : LREAL; (* Setpoint position *)
SetVelo               : LREAL; (* Setpoint velocity *)
SetAcc                : LREAL; (* Setpoint acceleration *)
TargetPos             : LREAL; (* Estimated target position *)
ModuloSetPos          : LREAL; (* Setpoint modulo position *)
ModuloSetTurns        : DINT;  (* Setpoint modulo turns *)
CmdNo                 : WORD;  (* Continuous actual command number *)
CmdState              : WORD;  (* Command state *)
END_STRUCT
END_TYPE
```

Erweiterte Beschreibung der Datenstruktur TYPE NCTOPLC_AXLESTRUCT2

8.1.3 Datentyp PLCTONC_AXIS_REF

Die Datenstruktur PLCTONC_AXIS_REF ist Bestandteil der AXIS_REF [▶ 105] Datenstruktur und übermittelt zyklisch Informationen an die NC. PLCTONC_AXIS_REF wird auch als Achsinterface zwischen SPS und NC bezeichnet.

```
TYPE PLCTONC_AXIS_REF
STRUCT
ControlDWord       : DWORD; (* Control double word *)
Override           : DWORD; (* Velocity override *)
AxisModeRequest    : DWORD; (* Axis operating mode (PLC request) *)
AxisModeDWord      : DWORD; (* optional mode parameter *)
AxisModeLReal      : LREAL; (* optional mode parameter *)
PositionCorrection : LREAL; (* Correction value for current position *)
ExtSetPos          : LREAL; (* external position setpoint *)
ExtSetVelo         : LREAL; (* external velocity setpoint *)
ExtSetAcc          : LREAL; (* external acceleration setpoint *)
ExtSetDirection    : DINT; (* external direction setpoint *)
Reserved1          : DWORD; (* reserved *)
ExtControllerOutput: LREAL; (* external controller output *)
GearRatio1         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
GearRatio2         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
GearRatio3         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
GearRatio4         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
MapState           : BYTE; (* reserved - internal use *)
Reserved_HIDDEN    : ARRAY [105..127] OF BYTE;
```

```
END_STRUCT
END_TYPE
```

Erweiterte Beschreibung der Datenstruktur TYPE PLCTONC_AXLESTRUCT

8.2 Motion Bausteine

8.2.1 Datentyp MC_BufferMode

Der Datentyp *MC_BufferMode* wird mit verschiedenen Funktionsbausteinen der Motion Control Bibliothek verwendet. Über BufferMode wird festgelegt, wie aufeinanderfolgende Bewegungskommandos abgearbeitet werden sollen.

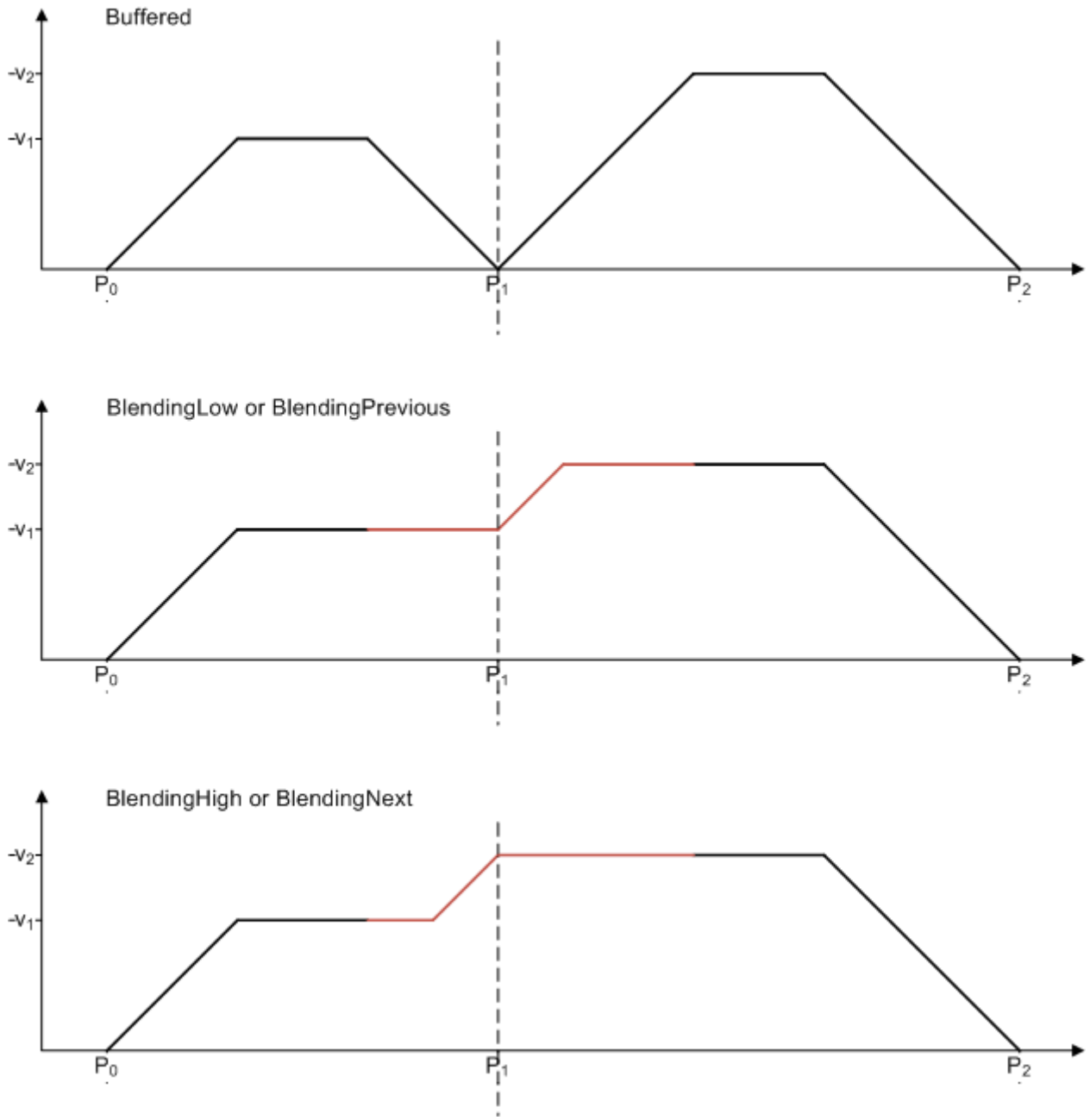
```
TYPE MC_BufferMode :
(
MC_Aborting,
MC_Buffered,
MC_BlendingLow,
MC_BlendingPrevious,
MC_BlendingNext,
MC_BlendingHigh
);
END_TYPE
```

siehe auch: BufferMode im Kapitel Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

Um den BufferMode zu verwenden, ist immer ein zweiter Funktionsbaustein nötigt. Es ist nicht möglich, einen Move-Baustein mit neuen Parametern zu triggern, während er noch aktiv ist.

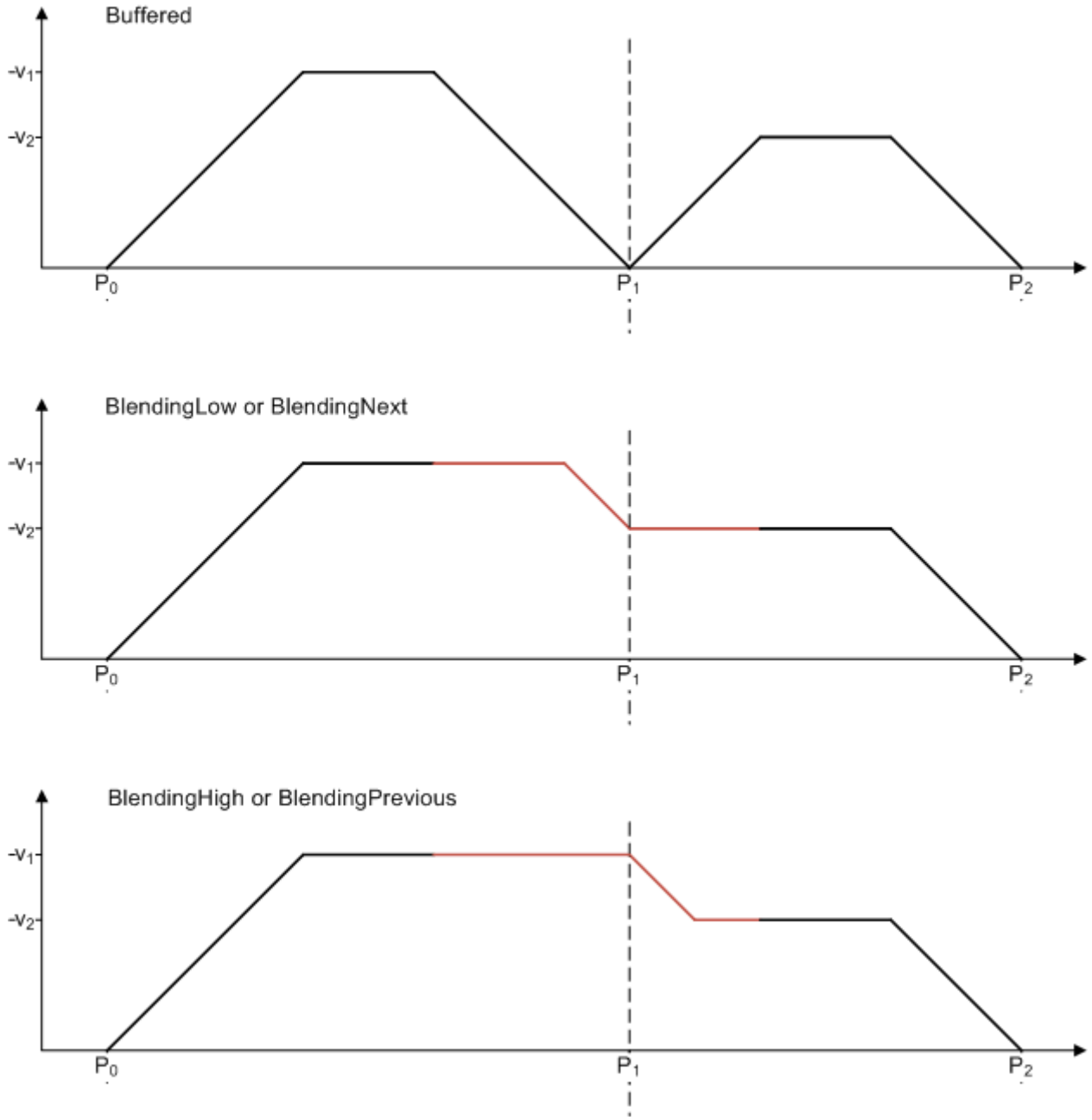
Beispiele:

Im folgenden Beispiel wird eine Achse mit zwei Move-Kommandos zunächst von Position P_0 auf P_1 und anschließend auf P_2 gefahren. Das zweite Kommando wird während der Fahrt nach P_1 aber noch vor der Bremsrampe mit verschieden Buffer-Modes beauftragt. Bezugspunkt für die verschiedenen Geschwindigkeitsprofile ist in jedem Fall P_1. Der Mode legt die Geschwindigkeit v_1 oder v_2 in diesem Punkt fest.



Da die Geschwindkeit des ersten Kommandos niedriger ist als die des zweiten, führen die Modes BlendingLow und BlendingPrevious bzw. die Modes BlendingHigh und BlendingNext jeweils zum selben Ergebnis.

Das nächste Beispiel unterscheidet sich dadurch, dass die Geschwindigkeit des zweiten Kommandos niedriger ist als die des ersten Kommandos. Hier sind nun die Modes BlendingLow und BlendingNext bzw. die Modes BlendingHigh und BlendingPrevious jeweils gleichwertig.



Die gezeigten Geschwindigkeitsprofile setzen voraus, dass das nachfolgende Kommando rechtzeitig, also noch vor der Bremsrampe des ersten Kommandos beauftragt wird. Anderenfalls wird das Blending bestmöglich umgesetzt.

8.2.2 Datentyp MC_Direction

```
TYPE MC_Direction :
(
MC_Positive_Direction := 1,
MC_Shortest_Way ,
MC_Negative_Direction,
MC_Current_Direction
);
END_TYPE
```

Dieser Aufzählungstyp enthält die möglichen Bewegungsrichtungen für die Funktionsbausteine MC_MoveVelocity [▶ 74] und MC_MoveModulo [▶ 66].

8.2.3 Datentyp MC_HomingMode

Der Datentyp *MC_HomingMode* zur Parametrierung des Funktionsbausteins MC_Home [▶ 92] verwendet

```
TYPE MC_HomingMode :
(
MC_DefaultHoming,     (* default homing as defined in the SystemManager encoder parameters *)
MC_AbsSwitch,         (* not implemented - Absolute Switch homing plus Limit switches *)
MC_LimitSwitch,       (* not implemented - Homing against Limit switches *)
MC_RefPulse,          (* not implemented - Homing using encoder Reference Pulse "Zero Mark" *)
MC_Direct,            (* Static Homing forcing position from user reference *)
MC_Absolute,          (* not implemented - Static Homing forcing position from absolute encoder *)
MC_Block,             (* not implemented - Homing against hardware parts blocking movement *)
MC_ForceCalibration, (* set the calibration flag without perfomring any motion or changing the
position *)
MC_ResetCalibration  (* resets the calibration flag without perfomring any motion or changing the
position *)
);
END_TYPE
```

8.2.4 Datentyp E_SuperpositionMode

```
TYPE E_SuperpositionMode :
(
SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION := 1,
SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION,
SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION,
SUPERPOSITIONMODE_LENGTHREDUCTION_LIMITEDMOTION,
SUPERPOSITIONMODE_ACCREDUCTION_ADDITIVEMOTION,   (* from TwinCAT 2.11 *)
SUPERPOSITIONMODE_ACCREDUCTION_LIMITEDMOTION     (* from TwinCAT 2.11 *)
);
END_TYPE
```

E_SuperpositionMode legt fest, wie eine überlagerte Bewegung mit dem Funktionsbaustein MC_MoveSuperImposed [▶ 85] durchgeführt wird.

Die mit *Veloreduction* benannten Modi führen die überlagerte Bewegung mit einer möglichst geringen Geschwindigkeitsänderung durch und nutzen bevorzugt die gesamte parametrierte Ausgleichstrecke. Umgekehrt nutzen die mit *Lengthreduction* benannten Modi die maximal mögliche Geschwindigkeitsänderung und verkürzen damit die benötigte Fahrstrecke. In beiden Fällen wird gleiche Distanz ausgeglichen.

Die in den mit *Additivemotion* benannten Fällen führt die überlagerte Achse eine längere oder kürzere Bewegung aus, als durch *Length* angegeben, da die überlagerte *Distanz* hinzukommt. Diese Modi werden beispielsweise verwendet, wenn sich der *Length* Parameter auf eine Vergleichsachse bezieht und die überlagerte Achse im Vergleich dazu eine längere oder kürzere Fahrtstrecke zurücklegen darf.

In den mit *Limitedmotion* benannten Fällen, wird die Überlagerung innerhalb der parametrierten Distanz abgeschlossen. Diese Modi werden beispielsweise verwendet, wenn sich der *Length* Parameter auf die überlagerte Achse selbst bezieht. Bei diesen Modi ist zu beachten, dass die überlagerte *Distanz* deutlich kürzer sein muss, als die zur Verfügung stehende Fahrtrecke *Length*.

SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION:

Die überlagerte Bewegung wird über die gesamte Strecke*Length* durchgeführt. Um die Distanz *Distance* auf dieser Strecke zu erreichen, wird die vorgegebene maximale GeschwindigkeitsänderungVelocityDiffreduziert.

Die Länge *Length* bezieht sich auf eine Vergleichs-Achse ohne überlagerte Bewegung (beispielsweise Master-Achse). Die Achse auf die die Ausgleichsfahrt wirkt, legt die Strecke *Length+Distance* zurück.

SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION:

Die überlagerte Bewegung wird über die gesamte StreckeLength durchgeführt. Um die Distanz *Distance* auf dieser Strecke zu erreichen, wird die vorgegebene maximale GeschwindigkeitsänderungVelocityDiffreduziert.

Die Länge *Length* bezieht sich auf die Achse, auf die die Ausgleichsfahrt wirkt. Diese legt während der Ausgleichsfahrt die Strecke *Length* zurück.

SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION:

Die überlagerte Bewegung wird auf möglichst kurzer Strecke mit möglichst hoher Geschwindigkeit ausgeführt. Dabei wird weder die maximale Geschwindigkeitsänderung *VelocityDiff* noch die maximale Strecke *Length* überschritten.

Die Länge *Length* bezieht sich auf eine Vergleichs-Achse ohne überlagerte Bewegung (beispielsweise Master-Achse). Die Achse auf die die Ausgleichsfahrt wirkt, legt maximal die Strecke *Length+Distance* zurück.

SUPERPOSITIONMODE_LENGTHREDUCTION_LIMITEDMOTION:

Die überlagerte Bewegung wird auf möglichst kurzer Strecke mit möglichst hoher Geschwindigkeit ausgeführt. Dabei wird weder die maximale Geschwindigkeitsänderung *VelocityDiff* noch die maximale Strecke *Length* überschritten.

Die Länge *Length* bezieht sich auf die Achse, auf die die Ausgleichsfahrt wirkt. Diese legt während der Ausgleichsfahrt maximal die Strecke *Length* zurück.

SUPERPOSITIONMODE_ACCREDUCTION_ADDITIVEMOTION (from TwinCAT 2.11)

Die überlagerte Bewegung wird über die gesamte Strecke*Length* durchgeführt. Um die Distanz *Distance* auf dieser Strecke zu erreichen, wird die vorgegebene maximale Beschleunigung *Acceleration* bzw. *Deceleration* soweit wie möglich reduziert.

Die Länge *Length* bezieht sich auf eine Vergleichs-Achse ohne überlagerte Bewegung (beispielsweise Master-Achse). Die Achse auf die die Ausgleichsfahrt wirkt, legt die Strecke *Length+Distance* zurück.

SUPERPOSITIONMODE_ACCREDUCTION_LIMITEDMOTION (from TwinCAT 2.11)

Die überlagerte Bewegung wird über die gesamte StreckeLength durchgeführt. Um die Distanz *Distance* auf dieser Strecke zu erreichen, wird die vorgegebene maximale Beschleunigung *Acceleration* bzw. *Deceleration* soweit wie möglich reduziert.

Die Länge *Length* bezieht sich auf die Achse, auf die die Ausgleichsfahrt wirkt. Diese legt während der Ausgleichsfahrt die Strecke *Length* zurück.

8.2.5 Datentyp ST_SuperpositionOptions

```
TYPE ST_SuperpositionOptions :
STRUCT
AbortOption : E_SuperpositionAbortOption;
END_STRUCT
END_TYPE

TYPE E_SuperpositionAbortOption :
(
SUPERPOSITIONOPTION_ABORTATSTANDSTILL := 0,
SUPERPOSITIONOPTION_RESUMEAFTERSTANDSTILL,
SUPERPOSITIONOPTION_RESUMEAFTERMOTIONSTOP
);
END_TYPE
```

AbortOption

AbortOption ist ein optionaler Parameter des Bausteins MC_MoveSuperimposed [▶ 85], der das Verhalten einer überlagerten Bewegung bei einem Stillstand der Hauptbewegung festlegt.

SUPERPOSITIONOPTION_ABORTATSTANDSTILL:

Die überlagerte Bewegung wird abgebrochen, sobald die unterlagerte Bewegung zu einem Stillstand der Achse führt. Einzige Ausnahme ist ein Stillstand, der durch einen Geschwindigkeits-Override von Null herbeigeführt wird. In diesem Fall wird auch die überlagerte Bewegung fortgesetzt, sobald der Override ungleich Null ist. *AbortAtStandstill* ist das Standardverhalten, falls die Option vom Anwender nicht belegt wird.

SUPERPOSITIONOPTION_RESUMEAFTERSTANDSTILL:

Die überlagerte Bewegung wird bei einem temporären Stillstand der Hauptbewegung nicht abgebrochen, sondern wird fortgesetzt, sobald sich die Achse wieder bewegt. Dieser Fall kann insbesondere bei einer Richtungsumkehr oder bei Kurvenscheibenbewegungen eintreten. Erst wenn die Zielposition der Achse erreicht ist oder die Achse gestoppt wurde, wird auch die überlagerte Bewegung beendet.

SUPERPOSITIONOPTION_RESUMEAFTERMOTIONSTOP:

Die überlagerte Bewegung wird bei einem Stillstand der Hauptbewegung nicht abgebrochen auch wenn die Achse ihre Zielposition erreicht hat oder gestoppt wurde. In diesem Fall wird die überlagerte Bewegung nach einem erneuten Start der Achse fortgesetzt.

Dieser Fall ist nicht von Bedeutung, falls die überlagerte Bewegung auf eine Slave-Achse angewendet wird, da diese nicht aktiv gestartet oder gestoppt werden kann. Bei Slave-Achsen sind die Betriebsarten *RESUMEAFTERSTANDSTILL* und *RESUMEAFTERMOTIONSTOP* gleichwertig. Die überlagerte Bewegung würde also auch nach einem erneuten Start der Master-Achse fortgesetzt.

Tab. 1: Übersicht über die Abbruchbedingungen einer überlagerten Bewegung (MC_MoveSuperimposed)

	ABORTATSTANDSTILL	RESUMEAFTERSTAND-STILL	RESUMEAFTERMOTI-ONSTOP
1. Override = 0%	wird fortgesetzt	wird fortgesetzt	wird fortgesetzt
2. temporärer Stillstand der Hauptbewegung	Abbruch	wird fortgesetzt	wird fortgesetzt
3. Bewegungsumkehr	Abbruch	wird fortgesetzt	wird fortgesetzt
4. Achse hat Zielposition erreicht oder wird gestoppt	Abbruch	Abbruch	wird fortgesetzt
5. Achs-Reset oder Abschalten des Enable-Signals	Abbruch	Abbruch	Abbruch
6. Bei Slave-Achsen: Abkoppeln	Abbruch	Abbruch	Abbruch

8.2.6 Datentyp E_JogMode

Der Datentyp E_JogMode wird in Verbindung mit dem Funktionsbaustein MC_Jog [▶ 94] verwendet.

```
TYPE E_JogMode :
(
MC_JOGMODE_STANDARD_SLOW, (* motion with standard jog parameters for slow motion *)
MC_JOGMODE_STANDARD_FAST, (* motion with standard jog parameters for fast motion *)
MC_JOGMODE_CONTINOUS,     (* axis moves as long as the jog button is pressed using parameterized
dynamics *)
MC_JOGMODE_INCHING,       (* axis moves for a certain relative distance *)
MC_JOGMODE_INCHING_MODULO (* axis moves for a certain relative distance - stop position is rounded
to the distance value *)
);
END_TYPE
```

8.3 Status und Parameter

8.3.1 Datentyp E_ReadMode

Der Datentyp E_ReadMode wird in Verbindung mit dem Funktionsbausteinen MC_ReadBoolParameter [▶ 30] und MC_ReadBoolParameter [▶ 28] verwendet, um einen einmaligen oder zyklischen Betrieb festzulegen.

```
TYPE E_ReadMode :
(
READMODE_ONCE := 1,
READMODE_CYCLIC
```

```
);
END_TYPE
```

8.3.2 Datentyp ST_AxisStatus

Der Datentyp *ST_AxisStatus* enthält umfangreiche Statusinformationen über eine Achse. Die Datenstruktur muss in jedem SPS-Zyklus durch einen Aufruf von MC_ReadStatus [▶ 32] oder durch einen Aufruf der Aktion Axis.ReadStatus (AXIS_REF [▶ 105]) aktualisiert werden.

```
TYPE ST_AxisStatus :
STRUCT
UpdateTaskIndex : BYTE; (* Task-Index of the task that updated this data set *)
UpdateCycleTime : LREAL; (* task cycle time of the task which calls the status function *)
CycleCounter          : UDINT; (* PLC cycle counter when this data set updated *)
NcCycleCounter        : UDINT; (* NC cycle counter incremented after NC task updated NcToPlc data
structures *)

MotionState           : MC_AxisStates; (* motion state in the PLCopen state diagram *)

Error                 : BOOL;  (* axis error state *)
ErrorId               : UDINT; (* axis error code *)

(* PLCopen motion control statemachine states: *)
ErrorStop             : BOOL;
Disabled              : BOOL;
Stopping              : BOOL;
StandStill            : BOOL;
DiscreteMotion        : BOOL;
ContinuousMotion      : BOOL;
SynchronizedMotion    : BOOL;
Homing                : BOOL;

(* additional status - (PLCopen definition)*)
ConstantVelocity      : BOOL;
Accelerating          : BOOL;
Decelerating          : BOOL;

(* Axis.NcToPlc.StateDWord *)
Operational           : BOOL;
ControlLoopClosed     : BOOL; (* operational and position control active *)
HasJob                : BOOL;
HasBeenStopped        : BOOL;
NewTargetPosition     : BOOL; (* new target position commanded during move *)
InPositionArea        : BOOL;
InTargetPosition      : BOOL;
Protected             : BOOL;
Homed                 : BOOL;
HomingBusy            : BOOL;
MotionCommandsLocked  : BOOL; (* stop 'n hold *)
SoftLimitMinExceeded  : BOOL; (* reverse soft travel limit exceeded *)
SoftLimitMaxExceeded  : BOOL; (* forward soft travel limit exceeded *)

Moving                : BOOL;
PositiveDirection     : BOOL;
NegativeDirection     : BOOL;
NotMoving             : BOOL;
Compensating          : BOOL; (* superposition - overlayed motion *)

ExtSetPointGenEnabled : BOOL;
ExternalLatchValid    : BOOL;
CamDataQueued         : BOOL;
CamTableQueued        : BOOL;
CamScalingPending     : BOOL;
CmdBuffered           : BOOL;
PTPmode               : BOOL;
DriveDeviceError      : BOOL;
IoDataInvalid         : BOOL;

(* Axis.NcToPlc.CoupleState *)
Coupled               : BOOL;

(* axis operation mode feedback from NcToPlc *)
OpMode                : ST_AxisOpModes;
END_STRUCT
```

```
END_TYPE
```

ST_AxisOpModes [▶ 117] MC_AxisStates [▶ 118]

8.3.3 Datentyp MC_AxisParameter

Der Datentyp *MC_AxisParameter* wird in Verbindung mit Funktionsbausteinen zum Lesen und Schreiben von Achs-Parametern verwendet.

```
TYPE MC_AxisParameter : (
(* PLCopen specific parameters *) (* Index-Group 0x4000 + ID*)
CommandedPosition := 1,           (* lreal *) (* taken from NcToPlc *)
SWLimitPos,                       (* lreal *) (* IndexOffset= 16#0001_000E *)
SWLimitNeg,                       (* lreal *) (* IndexOffset= 16#0001_000D *)
EnableLimitPos,                   (* bool  *) (* IndexOffset= 16#0001_000C *)
EnableLimitNeg,                   (* bool  *) (* IndexOffset= 16#0001_000B *)
EnablePosLagMonitoring,           (* bool  *) (* IndexOffset= 16#0002_0010 *)
MaxPositionLag,                   (* lreal *) (* IndexOffset= 16#0002_0012 *)
MaxVelocitySystem,                (* lreal *) (* IndexOffset= 16#0000_0027 *)
MaxVelocityAppl,                  (* lreal *) (* IndexOffset= 16#0000_0027 *)
ActualVelocity,                   (* lreal *) (* taken from NcToPlc *)
CommandedVelocity,                (* lreal *) (* taken from NcToPlc *)
MaxAccelerationSystem,            (* lreal *) (* IndexOffset= 16#0000_0101 *)
MaxAccelerationAppl,              (* lreal *) (* IndexOffset= 16#0000_0101 *)
MaxDecelerationSystem,            (* lreal *) (* IndexOffset= 16#0000_0102 *)
MaxDecelerationAppl,              (* lreal *) (* IndexOffset= 16#0000_0102 *)
MaxJerkSystem,                    (* lreal *) (* IndexOffset= 16#0000_0103 *)
MaxJerkAppl,                      (* lreal *) (* IndexOffset= 16#0000_0103 *)

(* Beckhoff specific parameters *) (* Index-Group 0x4000 + ID*)
AxisId := 1000,                   (* lreal *) (* IndexOffset= 16#0000_0001 *)
AxisVeloManSlow,                  (* lreal *) (* IndexOffset= 16#0000_0008 *)
AxisVeloManFast,                  (* lreal *) (* IndexOffset= 16#0000_0009 *)
AxisVeloMax,                      (* lreal *) (* IndexOffset= 16#0000_0027 *)
AxisAcc,                          (* lreal *) (* IndexOffset= 16#0000_0101 *)
AxisDec,                          (* lreal *) (* IndexOffset= 16#0000_0102 *)
AxisJerk,                         (* lreal *) (* IndexOffset= 16#0000_0103 *)
MaxJerk,                          (* lreal *) (* IndexOffset= 16#0000_0103 *)
AxisMaxVelocity,                  (* lreal *) (* IndexOffset= 16#0000_0027 *)
AxisRapidTraverseVelocity,        (* lreal *) (* IndexOffset= 16#0000_000A *)
AxisManualVelocityFast,           (* lreal *) (* IndexOffset= 16#0000_0009 *)
AxisManualVelocitySlow,           (* lreal *) (* IndexOffset= 16#0000_0008 *)
AxisCalibrationVelocityForward,   (* lreal *) (* IndexOffset= 16#0000_0006 *)
AxisCalibrationVelocityBackward,  (* lreal *) (* IndexOffset= 16#0000_0007 *)
AxisJogIncrementForward,          (* lreal *) (* IndexOffset= 16#0000_0018 *)
AxisJogIncrementBackward,         (* lreal *) (* IndexOffset= 16#0000_0019 *)
AxisEnMinSoftPosLimit,            (* bool  *) (* IndexOffset= 16#0001_000B *)
AxisMinSoftPosLimit,              (* lreal *) (* IndexOffset= 16#0001_000D *)
AxisEnMaxSoftPosLimit,            (* bool  *) (* IndexOffset= 16#0001_000C *)
AxisMaxSoftPosLimit,              (* lreal *) (* IndexOffset= 16#0001_000E *)
AxisEnPositionLagMonitoring,      (* bool  *) (* IndexOffset= 16#0002_0010 *)
AxisMaxPosLagValue,               (* lreal *) (* IndexOffset= 16#0002_0012 *)
AxisMaxPosLagFilterTime,          (* lreal *) (* IndexOffset= 16#0002_0013 *)
AxisEnPositionRangeMonitoring,    (* bool  *) (* IndexOffset= 16#0000_000F *)
AxisPositionRangeWindow,          (* lreal *) (* IndexOffset= 16#0000_0010 *)
AxisEnTargetPositionMonitoring,   (* bool  *) (* IndexOffset= 16#0000_0015 *)
AxisTargetPositionWindow,         (* lreal *) (* IndexOffset= 16#0000_0016 *)
AxisTargetPositionMonitoringTime, (* lreal *) (* IndexOffset= 16#0000_0017 *)
AxisEnInTargetTimeout,            (* bool  *) (* IndexOffset= 16#0000_0029 *)
AxisInTargetTimeout,              (* lreal *) (* IndexOffset= 16#0000_002A *)
AxisEnMotionMonitoring,           (* bool  *) (* IndexOffset= 16#0000_0011 *)
AxisMotionMonitoringWindow,       (* lreal *) (* IndexOffset= 16#0000_0028 *)
AxisMotionMonitoringTime,         (* lreal *) (* IndexOffset= 16#0000_0012 *)
AxisDelayTimeVeloPosition,        (* lreal *) (* IndexOffset= 16#0000_0104 *)
AxisEnLoopingDistance,            (* bool  *) (* IndexOffset= 16#0000_0013 *)
AxisLoopingDistance,              (* lreal *) (* IndexOffset= 16#0000_0014 *)
AxisEnBacklashCompensation,       (* bool  *) (* IndexOffset= 16#0000_002B *)
AxisBacklash,                     (* lreal *) (* IndexOffset= 16#0000_002C *)
AxisEnDataPersistence,            (* bool  *) (* IndexOffset= 16#0000_0030 *)
AxisRefVeloOnRefOutput,           (* lreal *) (* IndexOffset= 16#0003_0101 *)
AxisOverrideType,                 (* lreal *) (* IndexOffset= 16#0000_0105 *)
(* new since 4/2007 *)
AxisEncoderScalingFactor,         (* lreal *) (* IndexOffset= 16#0001_0006 *)
AxisEncoderOffset,                (* lreal *) (* IndexOffset= 16#0001_0007 *)
AxisEncoderDirectionInverse,      (* bool  *) (* IndexOffset= 16#0001_0008 *)
```

```
AxisEncoderMask,                    (* dword *) (* IndexOffset= 16#0001_0015 *)
AxisEncoderModuloValue,             (* lreal *) (* IndexOffset= 16#0001_0009 *)
AxisModuloToleranceWindow,          (* lreal *) (* IndexOffset= 16#0001_001B *)
AxisEnablePosCorrection,            (* bool  *) (* IndexOffset= 16#0001_0016 *)
AxisPosCorrectionFilterTime,        (* lreal *) (* IndexOffset= 16#0001_0017 *)
(* new since 1/2010 *)
AxisUnitInterpretation,             (* lreal *) (* IndexOffset= 16#0000_0026 *)
AxisMotorDirectionInverse,          (* bool  *) (* IndexOffset= 16#0003_0006 *)
(* new since 1/2011 *)
AxisCycleTime,                      (* lreal *) (* IndexOffset= 16#0000_0004 *)
(* new since 5/2011 *)
AxisFastStopSignalType,             (* dword *) (* IndexOffset= 16#0000_001E *)
AxisFastAcc,                        (* lreal *) (* IndexOffset= 16#0000_010A *)
AxisFastDec,                        (* lreal *) (* IndexOffset= 16#0000_010B *)
AxisFastJerk,                       (* lreal *) (* IndexOffset= 16#0000_010C *)

(* Beckhoff specific axis status information - READ ONLY *) (* Index-Group 0x4100 + ID*)
AxisTargetPosition := 2000,         (* lreal *) (* IndexOffset= 16#0000_0013 *)
AxisRemainingTimeToGo,              (* lreal *) (* IndexOffset= 16#0000_0014 *)
AxisRemainingDistanceToGo,          (* lreal *) (* IndexOffset= 16#0000_0022, 16#0000_0042 *)

(* Beckhoff specific axis functions *)
(* read/write gear ratio of a slave *)
AxisGearRatio := 3000,              (* lreal *) (* read: IndexGroup=0x4100+ID, IdxOffset=16#0000_0022,
*)
(* write:IndexGroup=0x4200+ID, IdxOffset=16#0000_0042 *)

(* Beckhoff specific other parameters *)
(* new since 1/2011 *)
NcSafCycleTime := 4000,             (* lreal *) (* IndexOffset= 16#0000_0010 *)
NcSvbCycleTime                      (* lreal *) (* IndexOffset= 16#0000_0012 *)
);
END_TYPE
```

Der Parameter *AxisGearRatio* kann nur gelesen oder geschrieben werden wenn die Achse als Slave gekoppelt ist. Während der Fahrt sind nur sehr kleine Änderungen erlaubt.

8.3.4 Datentyp ST_PowerStepperStruct

```
TYPE ST_PowerStepperStruct :
STRUCT
DestallDetectMode : E_DestallDetectMode;
DestallMode : E_DestallMode;
DestallEnable : BOOL;
StatusMonEnable : BOOL;
Retries : INT;
Timeout : TIME;
END_STRUCT
END_TYPE
```

8.3.5 Datentyp ST_DriveAddress

Der Datentyp *ST_DriveAddress* enthält die ADS Zugriffdaten eines Antriebsgerätes. Die Daten werden mit MC_ReadDriveAddress [▶ 58] gelesen.

```
TYPE ST_DriveAddress :
STRUCT
NetID        : T_AmsNetId;      (* AMS NetID of the drive as a string *)
NetIdBytes   : T_AmsNetIdArr;   (* AMS NetID of the drive as a byte array (same information as NetID)
*)
SlaveAddress : T_AmsPort;       (* slave address of the drive connected to a bus master *)
Channel      : BYTE;            (* EtherCAT channel number of the drive (0, 1, 2, 3, 4…) *)
END_STRUCT
END_TYPE
```

8.3.6 Datentyp ST_AxisParameterSet

Der Datentyp *ST_AxisParameterSet* enthält den gesamten Parameter-Datensatz einer Achse, der mit dem Funktionsbaustein MC_ReadParameterSet [▶ 31] gelesen werden kann.

Einzelne Parameter, die zur Laufzeit änderbar sind, können mit MC_WriteParameter [▶ 35] geschrieben werden. Es ist nicht möglich, den Parameterdatensatz als Ganzes zurückzuschreiben.

Die einzelnen Parameter sind in der NC-ADS-Dokumentation beschrieben.

```
TYPE ST_AxisParameterSet :
STRUCT
(* AXIS: *)
AxisId                      : DWORD; (* 0x00000001 *)
sAxisName                   : STRING(31); (* 0x00000002 *)
nAxisType                   : DWORD;      (* 0x00000003 *)
bEnablePositionAreaControl  : WORD;       (* 0x0000000F *)
fPositionAreaControlRange   : LREAL;      (* 0x00000010 *)
bEnableMotionControl        : WORD;       (* 0x00000011 *)
fMotionControlTime          : LREAL;      (* 0x00000012 *)
bEnableLoop                 : WORD;       (* 0x00000013 *)
fLoopDistance               : LREAL;      (* 0x00000014 *)
bEnableTargetPosControl     : WORD;       (* 0x00000015 *)
fTargetPosControlRange      : LREAL;      (* 0x00000016 *)
fTargetPosControlTime       : LREAL;      (* 0x00000017 *)
fVeloMaximum                : LREAL;      (* 0x00000027 *)
fMotionControlRange         : LREAL;      (* 0x00000028 *)
bEnablePEHTimeControl       : WORD;       (* 0x00000029 *)
fPEHControlTime             : LREAL;      (* 0x0000002A *)
bEnableBacklashCompensation : WORD;       (* 0x0000002B *)
fBacklash                   : LREAL;      (* 0x0000002C *)
sAmsNetId                   : T_AmsNetId; (* 0x00000031 *)
nPort                       : WORD;       (* 0x00000031 *)
nChnNo                      : WORD;       (* 0x00000031 *)
fAcceleration               : LREAL;      (* 0x00000101 *)
fDeceleration               : LREAL;      (* 0x00000102 *)
fJerk                       : LREAL;      (* 0x00000103 *)

(* ENCODER: *)
nEncId                      : DWORD;      (* 0x00010001 *)
sEncName                    : STRING(31); (* 0x00010002 *)
nEncType                    : DWORD;      (* 0x00010003 *)
fEncScaleFactor             : LREAL;      (* 0x00010006 *)
fEncOffset                  : LREAL;      (* 0x00010007 *)
bEncIsInverse               : WORD;       (* 0x00010008 *)
fEncModuloFactor            : LREAL;      (* 0x00010009 *)
nEncMode                    : DWORD;      (* 0x0001000A *)
bEncEnableSoftEndMinControl : WORD;       (* 0x0001000B *)
bEncEnableSoftEndMaxControl : WORD;       (* 0x0001000C *)
fEncSoftEndMin              : LREAL;      (* 0x0001000D *)
fEncSoftEndMax              : LREAL;      (* 0x0001000E *)
nEncMaxIncrement            : DWORD;      (* 0x00010015 *)
bEncEnablePosCorrection     : WORD;       (* 0x00010016 *)
fEncPosCorrectionFilterTime : LREAL;      (* 0x00010017 *)

(* CONTROLLER: *)
nCtrlId                     : DWORD;      (* 0x00020001 *)
sCtrlName                   : STRING(31); (* 0x00020002 *)
nCtrlType                   : DWORD;      (* 0x00020003 *)
bCtrlEnablePosDiffControl   : WORD;       (* 0x00020010 *)
bCtrlEnableVeloDiffControl  : WORD;       (* 0x00020011 *)
fCtrlPosDiffMax             : LREAL;      (* 0x00020012 *)
fCtrlPosDiffMaxTime         : LREAL;      (* 0x00020013 *)
fCtrlPosKp                  : LREAL;      (* 0x00020102 *)
fCtrlPosTn                  : LREAL;      (* 0x00020103 *)
fCtrlPosTv                  : LREAL;      (* 0x00020104 *)
fCtrlPosTd                  : LREAL;      (* 0x00020105 *)
fCtrlPosExtKp               : LREAL;      (* 0x00020106 *)
fCtrlPosExtVelo             : LREAL;      (* 0x00020107 *)
fCtrlAccKa                  : LREAL;      (* 0x00020108 *)

(* DRIVE: *)
nDriveId                    : DWORD;      (* 0x00030001 *)
sDriveName                  : STRING(31); (* 0x00030002 *)
nDriveType                  : DWORD;      (* 0x00030003 *)
bDriveIsInverse             : WORD;       (* 0x00030006 *)
fDriveVeloReferenz          : LREAL;      (* 0x00030101 *)
fDriveOutputReferenz        : LREAL;      (* 0x00030102 *)
```

```
(* miscellaneous *)
fAxisCycleTime              : LREAL;      (* 0x00000004 *) (* available from Tc 2.11 R2 *)

(* 17.05.11: parameter extension *)
fRefVeloSearch              : LREAL;      (* 0x00000006 calibration velo (TO plc cam) *)
fRefVeloSync                : LREAL;      (* 0x00000007 calibration velo (off plc cam) *)
fVeloSlowManual             : LREAL;      (* 0x00000008 manual velocity (slow) *)
fVeloFastManual             : LREAL;      (* 0x00000009 manual velocity (fast) *)

(* ENCODER (incremental): *)
bEncRefSearchInverse        : UINT;       (* 0x00010101 *)
bEncRefSyncInverse          : UINT;       (* 0x00010102 *)
nEncRefMode                 : UDINT;      (* 0x00010107 *)
fEncRefPosition             : LREAL;      (* 0x00010103 *)

(* fill up *)
arrReserved                 : ARRAY[511..512] OF BYTE; (* fill up to 512 bytes *)
END_STRUCT
END_TYPE
```

8.3.7 Datentyp ST_AxisOpModes

The data type *ST_AxisOpModes* contains information about the operating mode parameterisation of an axis.

```
TYPE ST_AxisOpModes :
STRUCT
PositionAreaMonitoring  : BOOL; (* bit 0 - OpModeDWord *)
TargetPositionMonitoring: BOOL; (* bit 1 - OpModeDWord *)
LoopMode                : BOOL; (* bit 2 - OpModeDWord - loop mode for two speed axes *)
MotionMonitoring        : BOOL; (* bit 3 - OpModeDWord *)
PEHTimeMonitoring       : BOOL; (* bit 4 - OpModeDWord *)
BacklashCompensation    : BOOL; (* bit 5 - OpModeDWord *)
Modulo                  : BOOL; (* bit 7 - OpModeDWord - axis is parameterized as modulo axis *)
PositionLagMonitoring   : BOOL; (* bit 16 - OpModeDWord *)
VelocityLagMonitoring   : BOOL; (* bit 17 - OpModeDWord *)
SoftLimitMinMonitoring  : BOOL; (* bit 18 - OpModeDWord *)
SoftLimitMaxMonitoring  : BOOL; (* bit 19 - OpModeDWord *)
PositionCorrection : BOOL; (* bit 20 - OpModeDWord *)
END_STRUCT
END_TYPE
```

8.3.8 Datentyp E_AxisPositionCorrectionMode

```
TYPE E_PositionCorrectionMode:
(
POSITIONCORRECTION_MODE_UNLIMITED, (* no limitation - pass correction immediately *)
POSITIONCORRECTION_MODE_FAST,      (* limitatation to maximum position change per cycle *)
POSITIONCORRECTION_MODE_FULLLENGTH (* limitation uses full length to adapt to correction in small
steps *)
);
END_TYPE
```

POSITIONCORRECTION_MODE_UNLIMITED	Keine Filterung, die Korrektur wird unmittelbar ausgeführt. Es ist zu beachten, das große Korrekturwertänderungen zu hohen Beschleunigungen führen können.
POSITIONCORRECTION_MODE_FAST	Die Positionskorrektur wird soweit begrenzt, dass eine maximale Beschleunigung nicht überschritten wird. Die Korrektur wird aber möglichst schnell vollständig ausgeführt.
POSITIONCORRECTION_MODE_FULLLENGTH	Die Positionskorrektur wird über eine Fahrstrecke der Achse (*CorrectionLength*) verteilt durchgeführt. Dadurch ergeben sich kleinere Änderungen pro Zeiteinheit.

8.3.9 Datentyp MC_AxisStates

Der Datentyp *MC_AxisStates* beschreibt die Betriebszustände nach dem PlcOpen Zustandsdiagramm [▶ 12].

```
TYPE MC_AxisStates :
(
MC_AXISSTATE_UNDEFINED,
MC_AXISSTATE_DISABLED,
MC_AXISSTATE_STANDSTILL,
MC_AXISSTATE_ERRORSTOP,
MC_AXISSTATE_STOPPING,
MC_AXISSTATE_HOMING,
MC_AXISSTATE_DISCRETEMOTION,
MC_AXISSTATE_CONTINOUSMOTION,
MC_AXISSTATE_SYNCHRONIZEDMOTION
);
END_TYPE
```

Siehe auch: Allgemeine Regeln für MC-Funktionsbausteine [▶ 15]

8.4 Touch Probe

8.4.1 Datentyp TRIGGER_REF

```
TYPE TRIGGER_REF :
STRUCT
EncoderID       : UDINT;              (* 1..255 *)
TouchProbe      : E_TouchProbe;       (* probe unit definition *)
SignalSource    : E_SignalSource;     (* optional physical signal source used by the probe unit
                                         - available from TwinCAT 2.11 Build 2022 with
MC_TouchProbe_V2 *)
Edge            : E_SignalEdge;       (* rising or falling signal edge *)
Mode            : E_TouchProbeMode;   (* single shot or continuous monitoring
                                         - available from TwinCAT 2.11 Build 2022 with
MC_TouchProbe_V2 *)
PlcEvent        : BOOL;               (* PLC trigger signal input when TouchProbe signal source is
set to 'PlcEvent' *)
ModuloPositions : BOOL;               (* interpretation of FirstPosition, LastPosition and
RecordedPosition as modulo positions when TRUE *)
END_STRUCT
END_TYPE
```

EncoderID : Die ID des Encoders kann im TwinCAT SystemManager abgelesen werden.

TouchProbe : Definiert die verwendete Latch-Einheit (Probe-Unit) innerhalb der verwendeten Encoder-Hardware.

```
TYPE E_TouchProbe :
(
TouchProbe1 := 1,     (* 1st hardware probe unit with Sercos, CanOpen, KL5xxx and others *)
TouchProbe2,          (* 2nd probe unit - available from MC_TouchProbe_V2_00 *)
TouchProbe3,          (* currently not available *)
TouchProbe4,          (* currently not available *)
PlcEvent := 10        (* simple PLC signal TRUE/FALSE *)
);
END_TYPE
```

SignalSource : Definiert optional die Signalquelle, soweit diese über die Steuerung gewählt werden kann. In vielen Fällen wird die Signalquelle fest im Antrieb konfiguriert und sollte dann auf den Standardwert *SignalSource_Default* eingestellt sein. (Einstellmöglichkeit erst mit MC_TouchProbe_V2 [▶ 39] verfügbar)

```
TYPE E_SignalSource :
(
SignalSource_Default,           (* undefined or externally configured *)
SignalSource_Input1,            (* digital drive input 1 *)
SignalSource_Input2,            (* digital drive input 2 *)
SignalSource_Input3,            (* digital drive input 3 *)
SignalSource_Input4,            (* digital drive input 4 *)
```

```
SignalSource_ZeroPulse := 128,  (* encoder zero pulse *)
SignalSource_DriveDefined       (* defined by drive parameters - e. g. CAN object 0x60D0 *)
);
END_TYPE
```

Edge : Legt fest, ob die steigende oder fallende Flanke des Trigger-Signals ausgewertet wird.

```
TYPE E_SignalEdge :
(
RisingEdge,
FallingEdge
);
END_TYPE
```

Mode : Legt die Betriebsart der Latch-Einheit fest. Im Single-Mode wird nur die erste Flanke erfasst. Im Continuous-Mode wird jede Flanke für einen SPS-Zyklus signalisiert. (*Mode* nur verfügbar mit MC_TouchProbe_V2 [▶ 39])

```
TYPE E_TouchProbeMode :
(
TOUCHPROBEMODE_SINGLE_COMPATIBILITYMODE,   (* for TwinCAT 2.10 and 2.11 before Build 2022 - for use
with MC_TouchProbe as well *)
TOUCHPROBEMODE_SINGLE,                     (* multi probe interface - from 2.11 Build 2022 *)
TOUCHPROBEMODE_CONTINOUS                   (* multi probe interface - from 2.11 Build 2022 *)
);
END_TYPE
```

PlcEvent : Wenn die Signalquelle TouchProbe auf den Typ PlcEvent eingestellt ist, so führt eine steigende Flanke an dieser Variablen zum Aufzeichnen der aktuellen Achsposition. Das PlcEvent ist keine echte Latch-Funktion, sondern Zykluszeit-abhängig.

ModuloPositions : Wenn die Variable *ModuloPositions* FALSE ist, so wird die Achsposition in einem absoluten linearen Bereich von -∞ bis +∞ interpretiert. Die Positionen *FirstPosition*, *LastPosition* und *RecordedPosition* des Funktionsbausteins MC_TouchProbe [▶ 36] bzw. MC_TouchProbe_V2 [▶ 39] sind dann ebenfalls absolut.
Wenn ModuloPositions TRUE ist, werden alle Positionen modulo im Modulo-Bereich der verwendeten Achse interpretiert (z. B. 0..359.9999). Gleichzeitig bedeutet das, dass ein definiertes Trigger-Fenster sich zyklisch wiederholt.

8.4.2 Datentyp MC_TouchProbeRecordedData

```
TYPE MC_TouchProbeRecordedData :
STRUCT
    Counter             : LREAL;
    RecordedPosition    : LREAL;
    AbsolutePosition    : LREAL;
    ModuloPosition      : LREAL;
END_STRUCT
END_TYPE
```

Counter: Zähler, der anzeigt, wie viele gültige Flanken im letzten Zyklus erkannt wurden. Eine Erkennung mehrerer Flanken ist nur im Mode = TOUCHPROBEMODE_CONTINIOUS unter SERCOS / SOE implementiert und muss explizit von der Hardware unterstützt werden (z.B. AX5000).

RecordedPosition: Erfasste Achsposition zum Zeitpunkt des Trigger-Signals. Diese entspricht je nach Parametrierung der absoluten Achsposition oder der Modulo Achsposition.

AbsolutePosition: Erfasste absolute Achsposition zum Zeitpunkt des Trigger-Signals.

ModuloPosition: Erfasste Modulo-Achsposition zum Zeitpunkt des Trigger-Signals.

8.5 Externer Sollwertgenerator

8.5.1 Datentyp E_PositionType

```
TYPE E_PositionType :
(
POSITIONTYPE_ABSOLUTE  := 1, (* Absolute position *)
POSITIONTYPE_RELATIVE,       (* Relative position *)
POSITIONTYPE_MODULO    := 5  (* Modulo position *)
);
END_TYPE
```

9 Beispielprogramme

9.1 Beispielprogramme

PTP - Punkt zu Punkt Bewegung

Das Beispielprogramm verwaltet und bewegt eine Achse im PTP-Modus. Die Achse wird mit zwei Instanzen eines MC_MoveAbsolute-Funktionsbausteins in der gepufferten Betriebsart über mehrere Zwischenpositionen und Geschwindigkeitsebenen bewegt.

Das Beispielprogramm benötigt die Bibliothek TcMC2.lib und läuft vollständig im Simulationsmodus. Der Ablauf kann im TwinCAT Scope View mit beiliegender Konfiguration verfolgt werden.

Zum Speichern des Beispielprogramms hier klicken:

https://infosys.beckhoff.com/content/1031/tcplclibmc2/Resources/458496267.zip

Master-Slave Kopplung

Das Beispielprogramm koppelt zwei Achsen und verfährt sie gemeinsam. Die Slave-Achse wird während der Fahrt abgekoppelt und positioniert.

https://infosys.beckhoff.com/content/1031/tcplclibmc2/Resources/458499211.zip

Tänzerregelung (Dancer Control)

Das Beispielprogram Dancer Control zeigt, wie die Geschwindigkeit einer Slave-Achse anhand der Position eines Tänzers geregelt werden kann.

https://infosys.beckhoff.com/content/1031/tcplclibmc2/Resources/458502155.zip

Überlagerte Bewegung (Superposition)

Das Beispiel zeigt die Überlagerung einer Bewegung während der Fahrt einer Achse.

https://infosys.beckhoff.com/content/1031/tcplclibmc2/Resources/458505099.zip

Mehr Informationen:
www.beckhoff.de/tx1200

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com