SIEMENS

SIMATIC

S7-1500
S7-1500T Interpreter functions
V10.0 as of STEP 7 V21

Function Manual

S7-1500T Motion Control

11/2025
A5E53131966-AC

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified persons are those who, because of their training and experience, are familiar with the installation, assembly, commissioning, operation, decommissioning and disassembly of the product and can recognize risks and avoid possible hazards.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the application described in the catalog and the associated usage information. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens Aktiengesellschaft. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens Aktiengesellschaft
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E53131966-AC
Ⓟ 09/2025 Subject to change

Table of contents

Introduction (S7-1500T)

1

Purpose of the documentation

This documentation provides important information that you need to configure and commission the integrated Motion Control functionality of the S7-1500 Automation systems.

Required basic knowledge

In order to understand this documentation, the following knowledge is required:

- General knowledge in the field of automation
- General knowledge in the field of drive engineering and motion control

Validity of the documentation

This documentation is valid for the S7-1500 product range.

Conventions

- The path specifications in the project tree assume that the "Technology objects" folder is open in the subtree of the CPU. The "Technology object" placeholder represents the name of the technology object.

  Example: "Technology object > Configuration > Basic parameters".
- The <TO> placeholder represents the name set in tags for the respective technology object.

  Example: <TO>.Actor.Type
- This documentation contains pictures of the devices described. The pictures may differ in minor details from the devices supplied.

You should also observe the notes that are marked as follows:

NOTE

A note contains important information about the product described in the documentation, about the handling of the product, and about sections in this documentation demanding your particular attention.

Industry Mall

The Industry Mall is the catalog and ordering system of Siemens Aktiengesellschaft for automation and drive solutions on the basis of Totally Integrated Automation (TIA) and Totally Integrated Power (TIP).

You can find catalogs for all automation and drive products on the Internet (https://mall.industry.siemens.com).

1.1 S7-1500 Motion Control Documentation Guide (S7-1500T)

Product information

Please also note the supplementary information on the Motion Control documentation:

- Product information on the S7-1500/1500T Motion Control documentation
  https://support.industry.siemens.com/cs/ww/en/view/109794046
  (https://support.industry.siemens.com/cs/ww/en/view/109794046)

Documentation

The documentation of the Motion Control functions is divided into the following documents:

- S7-1500/S7-1500T Motion Control Overview
  https://support.industry.siemens.com/cs/en/view/109990070
  (https://support.industry.siemens.com/cs/ww/en/view/109990070)
  This document describes the innovations in the technology versions, how to upgrade the technology version, functions that are used for all technology objects, and the operational sequence of Motion Control applications.
- S7-1500/S7-1500T Motion Control alarms and error IDs
  https://support.industry.siemens.com/cs/ww/en/view/109990076
  (https://support.industry.siemens.com/cs/ww/en/view/109990076)
  This document describes the technology alarms of the technology objects and the error identifications of the Motion Control instructions.
- S7-1500/S7-1500T Axis functions
  https://support.industry.siemens.com/cs/en/view/109990072
  (https://support.industry.siemens.com/cs/ww/en/view/109990072)
  This document describes the drive and encoder connection and functions for single-axis movements.
- S7-1500/S7-1500T Synchronous operation functions
  https://support.industry.siemens.com/cs/ww/en/view/109990074
  (https://support.industry.siemens.com/cs/ww/en/view/109990074)
  This document describes gearing, velocity synchronous operation and camming as well as cross-PLC synchronous operation.
- S7-1500/S7-1500T Measuring input and output cam functions
  https://support.industry.siemens.com/cs/ww/en/view/109990075
  (https://support.industry.siemens.com/cs/ww/en/view/109990075)

This document describes the detection of the actual position via a measuring input and the output of switching signals via output cam or cam track.

- S7-1500T kinematics functions
  https://support.industry.siemens.com/cs/ww/en/view/109990073
  (https://support.industry.siemens.com/cs/ww/en/view/109990073)
  This document describes the control of kinematics with up to 6 interpolating axes.
- S7-1500T Interpreter functions
  https://support.industry.siemens.com/cs/ww/en/view/109990077
  (https://support.industry.siemens.com/cs/ww/en/view/109990077)
  This documentation describes the control of technology objects via an Interpreter program.

See also

Topic page "SIMATIC Technology - Motion Control: Overview and Important Links"
https://support.industry.siemens.com/cs/ww/en/view/109751049
(https://support.industry.siemens.com/cs/ww/en/view/109751049)

1.2 Function Manuals documentation guide (S7-1500T)

1.2.1 Information classes Function Manuals (S7-1500T)

The documentation for the SIMATIC S7-1500 automation system, for the 1513/1516pro-2 PN, SIMATIC Drive Controller CPUs based on SIMATIC S7-1500 and the SIMATIC ET 200MP, ET 200SP, ET 200AL and ET 200eco PN distributed I/O systems is arranged into three areas.

This arrangement enables you to access the specific content you require.

You can download the documentation free of charge from the Internet (https://support.industry.siemens.com/cs/ww/en/view/109742705).

Basic information

The system manuals and Getting Started describe in detail the configuration, installation, wiring and commissioning of the SIMATIC S7-1500, SIMATIC Drive Controller, ET 200MP, ET 200SP, ET 200AL and ET 200eco PN systems. Use the corresponding operating instructions for 1513/1516pro-2 PN CPUs.

The STEP 7 online help supports you in the configuration and programming.

Examples:

- Getting Started S7-1500
- System manuals
- Operating instructions ET 200pro and 1516pro-2 PN CPU
- Online help TIA Portal

Device information

Equipment manuals contain a compact description of the module-specific information, such as properties, wiring diagrams, characteristics and technical specifications.

Examples:

- Equipment manuals for CPUs
- Equipment manuals for interface modules
- Equipment manuals for digital modules
- Equipment manuals for analog modules
- Equipment manuals for communication modules
- Equipment manuals for technology modules
- Equipment manuals for power supply modules
- Equipment manuals for BaseUnits

General information

The function manuals contain detailed descriptions on general topics relating to the SIMATIC Drive Controller and the S7-1500 automation system.

Examples:

- Function Manual Diagnostics
- Function Manual Communication
- Function Manuals Motion Control
- Function Manual Web Server
- Function Manual Cycle and Response Times
- PROFINET Function Manual
- PROFIBUS Function Manual

Product Information

Changes and supplements to the manuals are documented in a Product Information. The Product Information takes precedence over the device and system manuals.

You will find the latest Product Information on the Internet:

- S7-1500/ET 200MP (https://support.industry.siemens.com/cs/de/en/view/68052815)
- SIMATIC Drive Controller (https://support.industry.siemens.com/cs/de/en/view/109772684/en)
- Motion Control (https://support.industry.siemens.com/cs/de/en/view/109794046/en)
- ET 200SP (https://support.industry.siemens.com/cs/de/en/view/73021864)
- ET 200eco PN (https://support.industry.siemens.com/cs/ww/en/view/109765611)

Manual Collections

The Manual Collections contain the complete documentation of the systems put together in one file.

You will find the Manual Collections on the Internet:

- S7-1500/ET 200MP/SIMATIC Drive Controller (https://support.industry.siemens.com/cs/ww/en/view/86140384)
- ET 200SP (https://support.industry.siemens.com/cs/ww/en/view/84133942)
- ET 200AL (https://support.industry.siemens.com/cs/ww/en/view/95242965)
- ET 200eco PN (https://support.industry.siemens.com/cs/ww/en/view/109781058)

1.2.2 Basic tools (S7-1500T)

Tools

The tools described below support you in all steps: from planning, over commissioning, all the way to analysis of your system.

TIA Selection Tool

The TIA Selection Tool tool supports you in the selection, configuration, and ordering of devices for Totally Integrated Automation (TIA).

As successor of the SIMATIC Selection Tools , the TIA Selection Tool assembles the already known configurators for automation technology into a single tool.

With the TIA Selection Tool , you can generate a complete order list from your product selection or product configuration.

You can find the TIA Selection Tool on the Internet. (https://support.industry.siemens.com/cs/ww/en/view/109767888)

SIMATIC Automation Tool

You can use the SIMATIC Automation Tool to perform commissioning and maintenance activities on various SIMATIC S7 stations as bulk operations independent of TIA Portal.

The SIMATIC Automation Tool offers a wide range of functions:

- Scanning of a PROFINET/Ethernet system network and identification of all connected CPUs
- Assignment of addresses (IP, subnet, Gateway) and device name (PROFINET device) to a CPU
- Transfer of the date and the programming device/PC time converted to UTC time to the module
- Program download to CPU
- RUN/STOP mode switchover
- CPU localization through LED flashing
- Reading out of CPU error information

- Reading the CPU diagnostic buffer
- Reset to factory settings
- Firmware update of the CPU and connected modules

You can find the SIMATIC Automation Tool on the Internet.
(https://support.industry.siemens.com/cs/ww/en/view/98161300)

PRONETA

SIEMENS PRONETA (PROFINET network analysis) is a commissioning and diagnostic tool for PROFINET networks. PRONETA Basic has two core functions:

- In the network analysis, you get an overview of the PROFINET topology. Compare a real configuration with a reference installation or make simple parameter changes, e.g. to the names and IP addresses of the devices.
- The "IO test" is a simple and rapid test of the wiring and the module configuration of a plant, including documentation of the test results.

You can find SIEMENS PRONETA Basic on the Internet:
(https://support.industry.siemens.com/cs/ww/en/view/67460624)

SIEMENS PRONETA Professional is a licensed product that offers you additional functions. It offers you simple asset management in PROFINET networks and supports operators of automation systems in automatic data collection/acquisition of the components used through various functions:

- The user interface (API) offers an access point to the automation cell to automate the scan functions using MQTT or a command line.
- With PROFIenergy diagnostics, you can quickly detect the current pause mode or the readiness for operation of devices that support PROFIenergy and change these as needed.
- The data record wizard supports PROFINET developers in reading and writing acyclic PROFINET data records quickly and easily without PLC and engineering.

You can find SIEMENS PRONETA Professional on the Internet.
(https://www.siemens.com/proneta-professional)

SINETPLAN

SINETPLAN, the Siemens Network Planner, supports you in planning automation systems and networks based on PROFINET. The tool facilitates professional and predictive dimensioning of your PROFINET installation as early as in the planning stage. In addition, SINETPLAN supports you during network optimization and helps you to exploit network resources optimally and to plan reserves. This helps to prevent problems in commissioning or failures during productive operation even in advance of a planned operation. This increases the availability of the production plant and helps improve operational safety.

The advantages at a glance

- Network optimization thanks to port-specific calculation of the network load
- Increased production availability thanks to online scan and verification of existing systems
- Transparency before commissioning through importing and simulation of existing STEP 7 projects
- Efficiency through securing existing investments in the long term and the optimal use of resources

You can find SINETPLAN on the Internet (https://new.siemens.com/global/en/products/automation/industrial-communication/profinet/sinetplan.html).

1.2.3 SIMATIC Technical Documentation (S7-1500T)

Additional SIMATIC documents will complete your information. You can find these documents and their use at the following links and QR codes.

The Industry Online Support gives you the option to get information on all topics. Application examples support you in solving your automation tasks.

Overview of the SIMATIC Technical Documentation

Here you will find an overview of the SIMATIC documentation available in Siemens Industry Online Support:

Industry Online Support International (https://support.industry.siemens.com/cs/ww/en/view/109742705)

Watch this short video to find out where you can find the overview directly in Siemens Industry Online Support and how to use Siemens Industry Online Support on your mobile device:

Quick introduction to the technical documentation of automation products per video (https://support.industry.siemens.com/cs/us/en/view/109780491)

YouTube video: Siemens Automation Products - Technical Documentation at a Glance (https://youtu.be/TwLSxxRQQsA)

Retention of the documentation

Retain the documentation for later use.

For documentation provided in digital form:

1. Download the associated documentation after receiving your product and before initial installation/commissioning. Use the following download options:
   - Industry Online Support International: (https://support.industry.siemens.com)

     The article number is used to assign the documentation to the product. The article number is specified on the product and on the packaging label. Products with new, non-compatible functions are provided with a new article number and documentation.
   - ID link:

     Your product may have an ID link. The ID link is a QR code with a frame and a black frame corner at the bottom right. The ID link takes you to the digital nameplate of your product. Scan the QR code on the product or on the packaging label with a smartphone camera, barcode scanner, or reader app. Call up the ID link.
2. Retain this version of the documentation.

Updating the documentation

The documentation of the product is updated in digital form. In particular in the case of function extensions, the new performance features are provided in an updated version.

1. Download the current version as described above via the Industry Online Support or the ID link.
2. Also retain this version of the documentation.

mySupport

With "mySupport" you can get the most out of your Industry Online Support.

Registration	You must register once to use the full functionality of "mySupport". After registration, you can create filters, favorites and tabs in your personal workspace.
Support requests	Your data is already filled out in support requests, and you can get an overview of your current requests at any time.
Documentation	In the Documentation area you can build your personal library.
Favorites	You can use the "Add to mySupport favorites" to flag especially interesting or frequently needed content. Under "Favorites", you will find a list of your flagged entries.
Recently viewed articles	The most recently viewed pages in mySupport are available under "Recently viewed articles".
CAx data	The CAx data area gives you access to the latest product data for your CAx or CAe system. You configure your own download package with a few clicks: • Product images, 2D dimension drawings, 3D models, internal circuit diagrams, EPLAN macro files • Manuals, characteristics, operating manuals, certificates • Product master data

You can find "mySupport" on the Internet. (https://support.industry.siemens.com/My/ww/en)

Application examples

The application examples support you with various tools and examples for solving your automation tasks. Solutions are shown in interplay with multiple components in the system - separated from the focus on individual products.

You can find the application examples on the Internet.
(https://support.industry.siemens.com/cs/ww/en/ps/ae)

Safety instructions (S7-1500T) 2

2.1 General safety instructions (S7-1500T)

The safety instructions can be found in section "Safety instructions" in the System Manual "SIMATIC S7-1500, ET 200MP Automation System" (https://support.industry.siemens.com/cs/ww/en/view/59191792) document.

Cybersecurity-relevant information can be found in the Industrial Cybersecurity (Page 19) section.

2.2 Warnings in this document (S7-1500T)

You can find explanations of the warnings used in this document in the "Legal information" section.

Industrial cybersecurity (S7-1500T) 3

Due to the digitalization and increasing networking of machines and industrial plants, the risk of cyber attacks is also growing. Appropriate protective measures are therefore mandatory, particularly in the case of critical infrastructure facilities.

3.1 Cybersecurity information (S7-1500T)

Siemens provides products and solutions with industrial cybersecurity functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial cybersecurity concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial cybersecurity measures that may be implemented, please visit
https://www.siemens.com/cybersecurity-industry.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Cybersecurity RSS Feed under
https://new.siemens.com/cert.

New features V10.0 (S7-1500T)

4

4.1 New Interpreter functions V10.0 (S7-1500T)

Technology version V10.0 contains the following new Interpreter functions:

Extensions of the Interpreter mapping

Mapping complex data types for describing position data is supported, e.g. TO_Struct_Ipr_Frame.

Mapping of arrays is supported.

Changing the values of tag in "Debug" program mode

In "Debug" program mode, you can display the value of a tag and change the value at the breakpoints, e.g. for test purposes.

Conversion rules for explicit data type conversion LREAL_TO_DINT or LREAL_TO_UDINT

The conversion of floating-point numbers into DINT/UDINT values now provides a defined result, even when values outside of the valid range or invalid floating-point numbers are entered. Previously, the result of the conversion was undefined in such cases.

Overview of functions (S7-1500T) 5

With the SIMATIC Motion Interpreter of the S7-1500T CPU, you create motion jobs for individual axes and kinematics with up to 6 interpolating axes.

The Interpreter executes a series of serial instructions. This sequence is called an Interpreter program.

Advantages

- Sequential programming of technology and motion tasks takes place separately and independently of the cyclic user program of the CPU
- Integrated extensions for technological tasks, e.g. path-synchronous actions
- Motion jobs are executed with support from the system in a technological and time-optimized manner with preparation and execution control.
- Fast, easy programming with configuration dialogs
- Quick and easy commissioning thanks to 3D visualization of the kinematics in the programming editor
- Reusability of Interpreter programs for similar technological tasks
- Calibration of positions

Motion Control Language

You describe diverse technological tasks textually for the Interpreter in the Interpreter language "Motion Control Language" (MCL):

- Support for SCL language constructs and data types
- Logical operations, operations with variables and mathematical functions
- Adaptation of the language to interpretative processing by the Interpreter
- Sequential programming in an Interpreter program
- Enabling, disabling, and homing of axes
- Easy programming of motion jobs on individual axes
- Enabling/disabling of force and torque limiting and fixed stop detection
- Easy programming of complex motion jobs on kinematics, e.g. for a pick-and-place cycle
- Linear, circular, and synchronous "point-to-point" motions with absolute and relative position specification
- Defining tool and object coordinate systems
- Defining, activating, and deactivating workspace zones and kinematics zones
- Setting modal parameters, such as dynamic parameters

- Synchronized actions, such as setting outputs, starting jobs on other technology objects, calculations
- System functions for controlling program execution and integration into the processes of the user program

In contrast to cyclic programming with STEP 7, you do not need to program any step enabling conditions or queries in MCL to check whether a job has been completed.

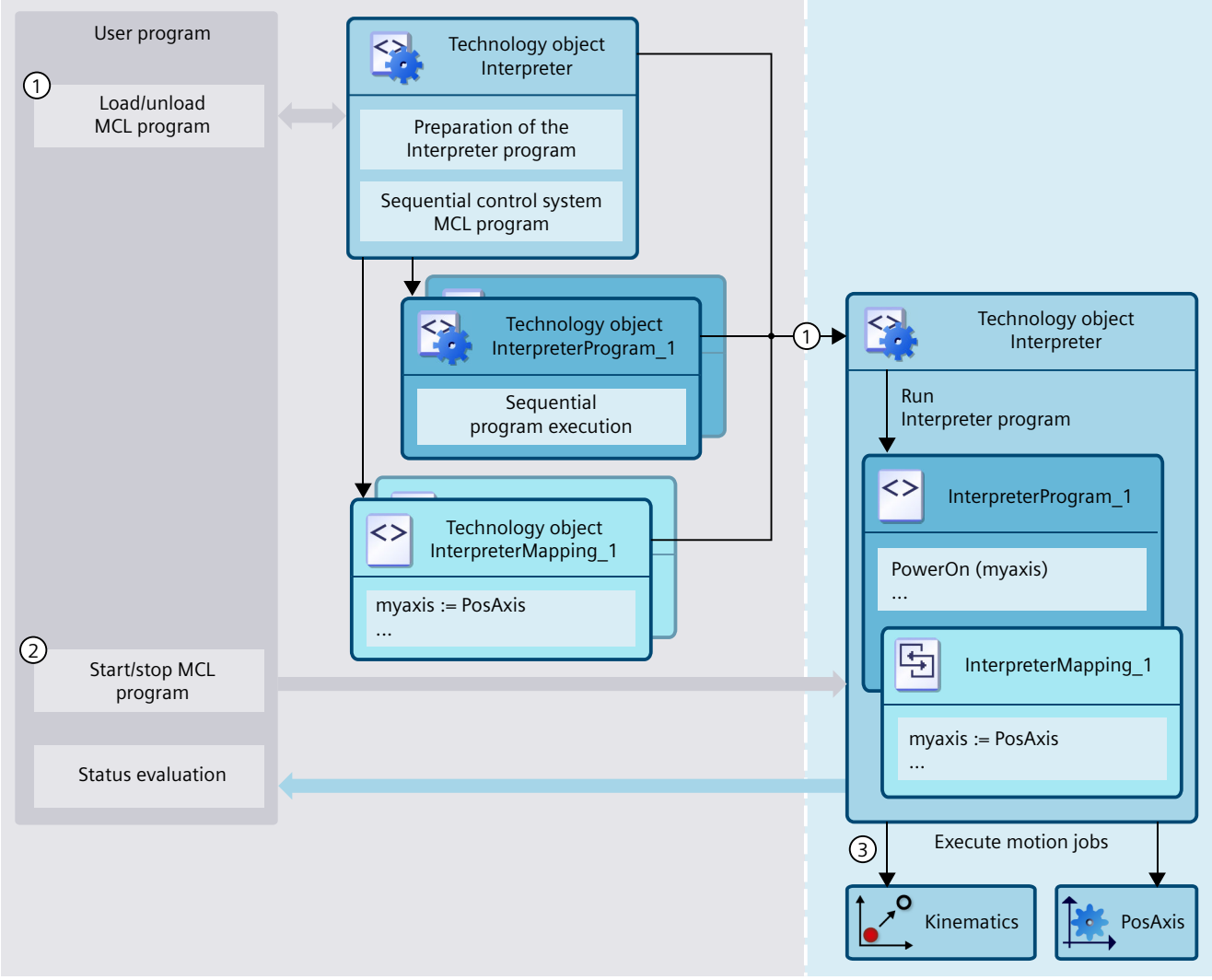
Principle of operation

The SIMATIC Motion Interpreter is implemented as an Interpreter technology object. The Interpreter technology object is controlled using instructions from the user program. The predefined clipboard of the Interpreter technology object enables the exchange of variables between the user program and the Interpreter program.

You create the Interpreter program in the Interpreter program technology object.

The Interpreter mapping technology object provides access to technology objects and PLC variables.

In the following figure, the motion sequence from the Interpreter program technology object is loaded from the user program into the Interpreter technology object ①. Processing of the motion jobs is started via the user program ②. The Interpreter executes the motion tasks on the Kinematics technology object and on the Positioning Axis technology object ③. The positioning axis is connected to the Interpreter via an Interpreter mapping.



Licenses

A license is required for the SIMATIC Motion Interpreter product.

To use the SIMATIC Motion Interpreter, you need a runtime license for each technology CPU.

Product	Article number
SIMATIC Motion Interpreter basic	6ES7823-0SJ00-1AA0

You can test the functions of the SIMATIC Motion Interpreter with SIMATIC S7-PLCSIM or SIMATIC S7-PLCSIM Advanced.

See also

Interpreter technology object (Page 24)
Interpreter program technology object (Page 27)
Interpreter mapping technology object (Page 28)

5.1 Interpreter technology object (S7-1500T)

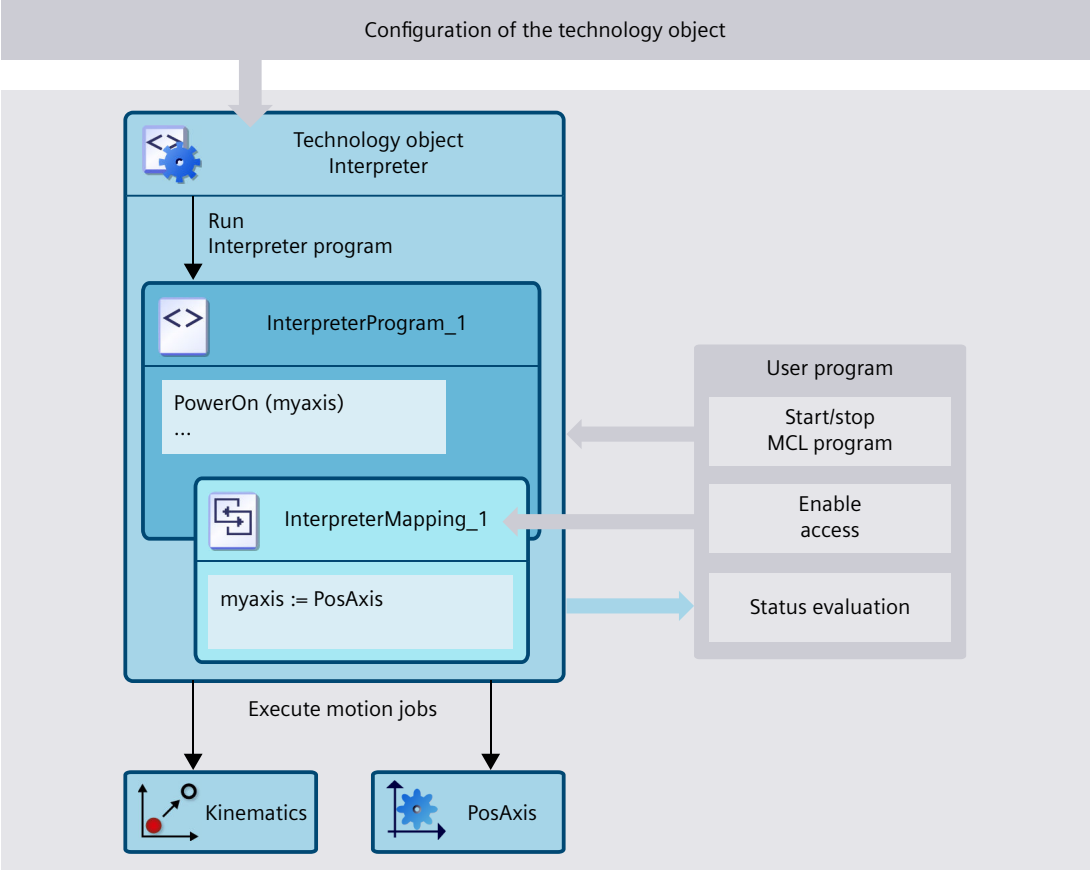
The interpreter technology object has the following tasks:

- Loading/unloading of an interpreter program
- Loading/unloading of an Interpreter mapping
- Preparation of the interpreter program
- Technology and time-optimized processing of the interpreter program
- Execution of motion jobs

The interpreter technology object is controlled from within the user program.

An interpreter technology object can control a kinematics technology object and its kinematics axes, as well as mapped Speed Axis, Positioning Axis, and Synchronous Axis technology objects.

In the following figure, the interpreter technology object processes InterpreterProgram_1. The interpreter technology object executes the motion jobs on the kinematics and on the additional PosAxis axis, which is mapped via InterpreterMapping_1.



Data exchange with the user program

The interpreter program executed by the interpreter technology object can access the following PLC data:

- Tags of the assigned kinematics technology object
- Tags of the interconnected kinematics axes of the assigned kinematics technology object
- Tags of the mapped technology objects
- Mapped tags of global data blocks

The user program of the CPU can access the following data of the interpreter technology object:

- Tags of the data block of the Interpreter technology object, e.g. configuration of the length of the Interpreter job sequence, clipboard tags, ErrorWord, StatusWord

Configuration

The following sections are relevant for configuring the interpreter technology object:

- Basic parameters
  - Connecting Interpreter to a kinematics technology object (Page 35)
- Extended parameters
  - Configuring the interpreter job sequence (Page 46)
  - Configuring the maximum wait time (Page 46)
  - Configuring the program override (Page 47)
  - Configuring clipboard tags (Page 40)

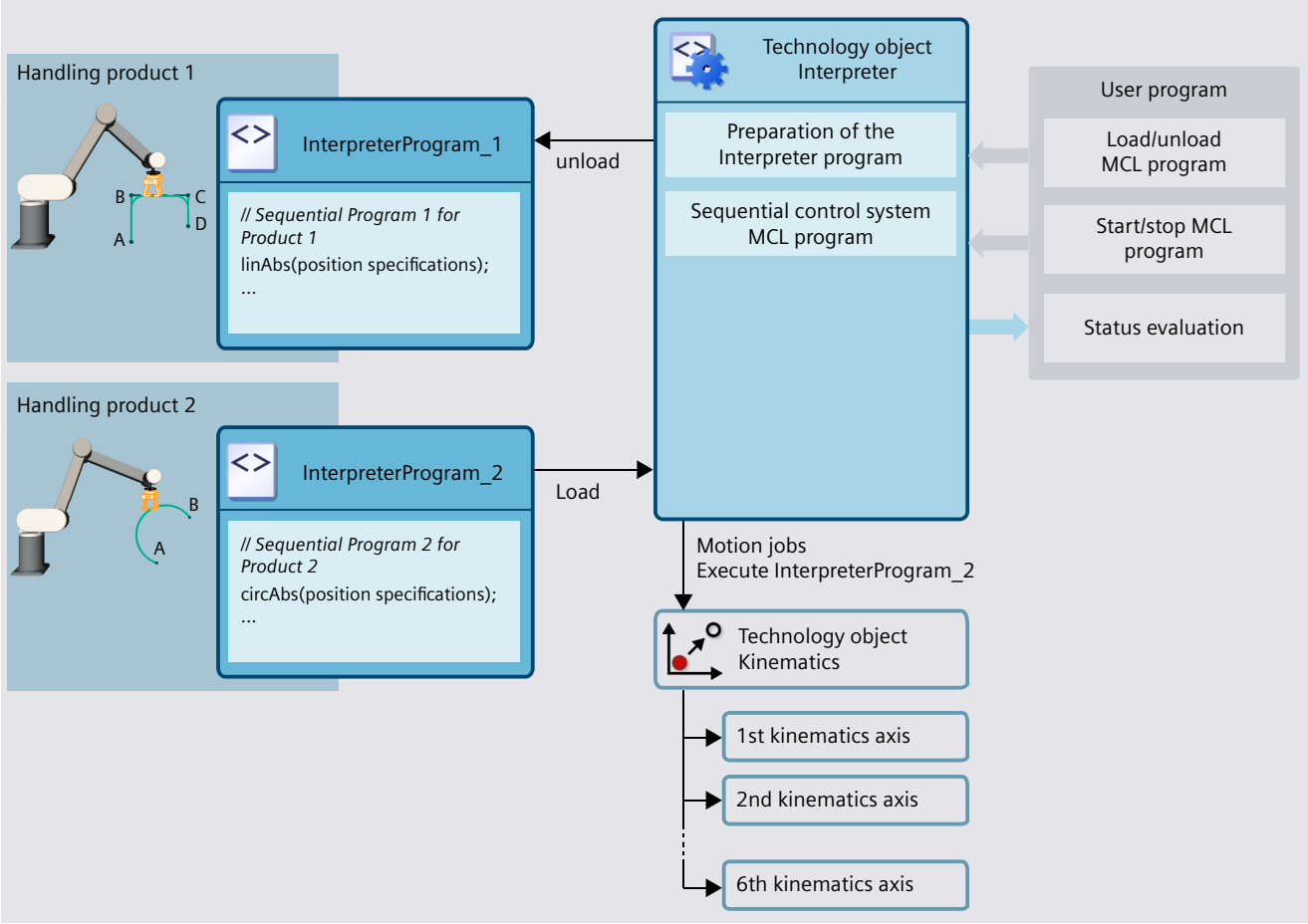
5.2 Interpreter program technology object (S7-1500T)

The Interpreter program contains the motion jobs for the kinematics and/or single axes. The Interpreter program is loaded into the Interpreter technology object during runtime and interpreted. The Interpreter program does not have to be compiled in the TIA Portal.

You create the Interpreter programs in the programming editor of the Interpreter program technology object.

The Interpreter program technology object has an editor for creating an Interpreter program in the Motion Control Language (MCL) programming language.

In the following figure, there are two Interpreter program technology objects. Each technology object contains a programmed motion sequence for the production of a product. Unloading one Interpreter program and loading the other Interpreter program changes the Interpreter program in the Interpreter technology object. The Interpreter then executes the motion sequence for producing the other product.



You can connect an Interpreter program technology object to multiple Interpreter technology objects. This allows you to control multiple kinematics with the same Interpreter program.

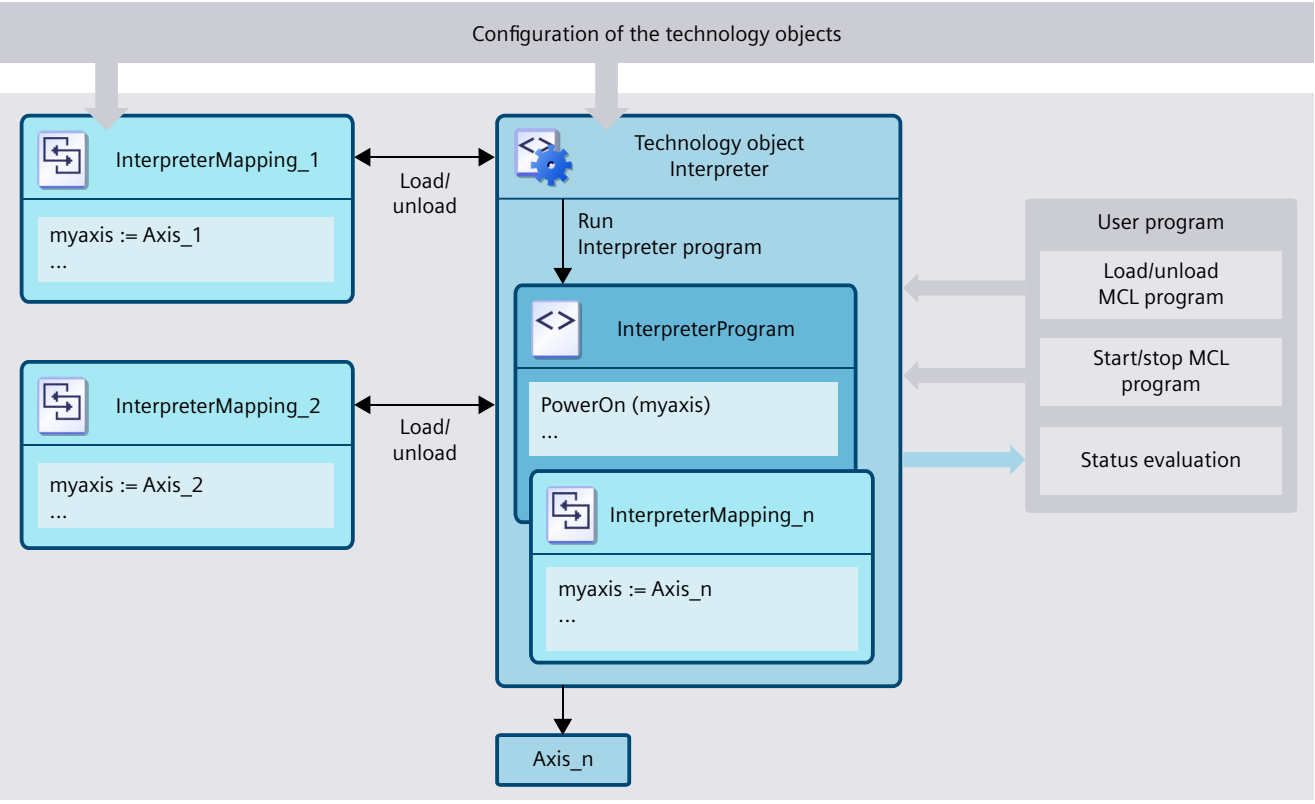
5.3 Interpreter mapping technology object (S7-1500T)

With the Interpreter mapping technology object you can define an assignment of objects in the Interpreter program to objects in the user program. This specifies which CPU objects you can access from an Interpreter program.

You can map the following CPU objects for access from the Interpreter program:

- Speed axis, positioning axis, and synchronous axis technology object types
- Variables of global data blocks

In the following figure, two Interpreter mapping technology objects are created. Unloading one Interpreter mapping and loading the other Interpreter mapping changes the Interpreter mapping in the Interpreter technology object. This allows the Interpreter to execute the motion sequence on a different axis.



Configuration

- Mapping of technology objects (Page 40)
- Mapping of variables (Page 41)

5.4 Motion Control instructions for interpreter control (S7-1500T)

The functions of the Interpreter technology object can be executed in the user program with the following Motion Control instructions:

- "MC_LoadProgram": Load/unload the Interpreter program
- "MC_RunProgram": Execution of the Interpreter program
- "MC_StopProgram": Stop execution of the Interpreter program

5.5 Functions in STEP 7 (S7-1500T)

The technology objects support the following functions in STEP 7:

- MCL programming language (Page 72)
- Programming editor (Page 166) with 3D visualization of the kinematics
- Diagnostics (Page 338): Monitor and observe status and error messages of the technology objects

5.6 MCL instructions (S7-1500T)

The following tables list the MCL instructions (Page 186) supported by the Interpreter program:

Kinematics motions

The following table lists the MCL instructions supported by the Interpreter program for kinematics motion (Page 186).

MCL instruction	Description
linAbs()	Position kinematics absolutely with linear path motion
linRel()	Relative positioning of kinematics with linear path motion
circAbs()	Position kinematics absolutely with circular path motion
circRel()	Relative positioning of kinematics with circular path motion
ptpAbs()	Move kinematics to absolute Cartesian target coordinates with sPTP motion
ptpRel()	Move kinematics to relative Cartesian target coordinates with sPTP motion
ptpAxAbs()	Move kinematics to absolute axis positions with sPTP motion
ptpAxRel()	Move kinematics to relative axis positions with sPTP motion
ptpJtAbs()	Move kinematics to absolute joint positions with sPTP motion
ptpJtRel()	Move kinematics to relative joint positions with sPTP motion
setDyn()	Set dynamic defaults for path motions modally
setOriDyn()	Set dynamic defaults for orientation motions modally
setPtpDyn()	Set dynamic defaults for sPTP motions modally

MCL instruction	Description
setDynMax()	Set dynamic limits for path motions modally
setOriDynMax()	Set dynamic limits for orientation motions modally
setTrans()	Set motion transition for linear, circular path and sPTP motions modally
setBlend()	Set blending mode for linear and circular path motions modally
setBlendDist()	Set blending distance for linear, circular path and sPTP motions modally
setBlendFactor()	Set max. blending distance for linear, circular path and sPTP motions modally
setPlane()	Set main plane of circle path for circular path motions modally
setOriDirA()	Set direction of motion of Cartesian orientation A manually Relevant for linAbs(), circAbs(), and ptpAbs()
setTurnJoint()	Set target joint position ranges for sPTP motions modally
setCircDir()	Set orientation of circle path for circular path motions modally
setLc()	Set target joint position space for sPTP motions modally
setDynAdapt()	Set dynamic adaptation for path motions modally

Coordinate systems

The following table lists the MCL instructions supported by the Interpreter program for coordinate systems (Page 272).

MCL instruction	Description
defOcs()	Redefine object coordinate systems
defTool()	Redefine tool
trackIn()	Start conveyor tracking
setCs()	Set reference coordinate system for linear, circular path and sPTP motions modally

Zones

The following table lists the MCL instructions supported by the Interpreter program for zones (Page 282).

MCL instruction	Description
defWsZone()	Define workspace zone
defKinZone()	Define kinematics zone
setWsZoneActive()	Activate workspace zone
setWsZoneInactive()	Deactivate workspace zone
setKinZoneActive()	Activate kinematics zone
setKinZoneInactive()	Deactivate kinematics zone

Tools

The following table lists the MCL instructions supported by the Interpreter program for tools (Page 291).

MCL instruction	Description
setTool()	Change active tool

Axis functions

The following table lists the MCL instructions supported by the Interpreter program for axis functions (Page 293).

MCL instruction	Description
move()	Move axis with velocity/speed setpoint
posAbs()	Position axis absolutely
posRel()	Position axis relatively
setAxisDyn()	Set dynamic defaults for single-axis motions modally
setAxisDynMax()	Set dynamic limits for single-axis motions modally
powerOn()	Enable axis
powerOff()	Disable axis
home()	Home axis, set reference point
torqueLimitOn()	Activate force/torque limit/fixed stop detection
torqueLimitOff()	Deactivate force/torque limit/fixed stop detection

Other instructions

The following table lists the MCL instructions supported by the Interpreter program for other instructions (Page 315).

MCL instruction	Description
writeVar()	Write mapped PLC tag or technology object data block tag
waitTime()	Interrupt program execution for a defined period
waitEvent()	Interrupt program execution until a specific event
setControlledByInterpreter()	Set "ControlledByInterpreter" bit for a technology object
setOvr()	Set program override
preHalt()	Halt program preparation

Control instructions

The following table lists the MCL instructions supported by the Interpreter program for control instructions (Page 122).

MCL instruction	Description
IF/ELSE	Either-or instruction
CASE	Conditional instruction
FOR	FOR loop
WHILE	WHILE loop
REPEAT	REPEAT-UNTIL loop
EXIT	Cancel loop unconditionally
CONTINUE	Cancel loop conditionally
GOTO	Jump to defined jump label
RETURN	Exit function/main program
SYNC ON_START	Start-triggered synchronous action
SYNC ON_POS	Position-triggered synchronous action

Bitstring instructions

The Interpreter program supports bitstring instructions.

Math functions

The Interpreter program supports mathematical functions.

Conversions

The Interpreter program supports conversions.

5.7 Configuration limits (S7-1500T)

General configuration

	Number
Maximum depth of the program preparation	Maximum 100 (configurable)
Maximum number jobs from kinematics instructions	30
Maximum number of mapped technology objects	20
Maximum number of characters per instruction	300
Maximum number of identifiers	80

Structure of Interpreter programs

Maximum number of functions or interrupt routines	50
Maximum call depth in program execution	8
Maximum nesting depth	50
Maximum number of user-defined data types	32
Maximum nesting depth for user-defined data types	8
Maximum number of labels	100

Number of variables

Maximum number of global variables in an Interpreter technology object	500
Maximum number of local variables and function parameters in an Interpreter technology object	50
Maximum number of mapped variables in an Interpreter mapping	100
Maximum dimension of user-defined arrays	1

Parallel execution of jobs and motions

Maximum number of programmable synchronous actions	10

Interpreter functions (S7-1500T) 6

6.1 Term definition (S7-1500T)

Motion sequence

A motion sequence refers to several motion jobs that are blended into one another. A motion job that is not blended with either the previous or the subsequent motion job is an independent motion sequence in its own right.

Program preparation

The Interpreter technology object interprets the jobs from the loaded Interpreter program by adding the jobs to the Interpreter sequence and preparing them. When the interpreter program is loaded and during program execution, the interpreter prepares the jobs in advance.

Program execution

During program execution, the already prepared jobs are executed by the interpreter technology object or the technology objects controlled by the interpreter.

Synchronized action

A synchronized action is a job that runs in parallel with other jobs, such as opening a gripper during a kinematics motion.

6.2 Connecting technology objects (S7-1500T)

6.2.1 Relationships between technology objects (S7-1500T)

Connecting an Interpreter technology object to a kinematics technology object

- An Interpreter technology object can be connected to exactly one kinematics technology object.
- You can access the kinematics axes of the connected kinematics from the Interpreter program.

Relationship between interpreter technology object and interpreter program technology object

- An interpreter technology object can have exactly one interpreter program loaded at a time.
- An interpreter program can be loaded in multiple interpreter technology objects.

Relationship between interpreter technology object and interpreter mapping technology object

- An interpreter technology object can have exactly one interpreter mapping loaded at a time, whereas
- one interpreter mapping can be loaded in multiple interpreter technology objects.

6.2.2 Connecting Interpreter to a kinematics (S7-1500T)

To execute the motion jobs of a kinematics, the Interpreter technology object must be connected to the kinematics technology object.

If the Interpreter technology object has been connected, a link to the Interpreter technology object appears in the kinematics technology object in the project tree.

Requirements

- An Interpreter technology object has been created.
- A Kinematics technology object has been created.

Procedure

You have the following options for connecting the Interpreter technology object to a kinematics technology object:

- When adding an "Interpreter" technology object in the "Add technology object" window, select the Kinematics technology object from the "Kinematics (optional)" drop-down list.
- In the project tree, drag-and-drop the Interpreter technology object onto the Kinematics technology object.
- In the configuration of the Interpreter technology object, select the kinematics technology object under "Basic parameters > Connected kinematics".

Result

A link to the connected Interpreter technology object is created in the kinematics technology object in the project tree.

Removing the connection

You have the following options for removing the connection of an Interpreter technology object to a kinematics technology object:

- Delete the link in the kinematics technology object.
- Remove the selection under "Basic parameters > Connected kinematics" in the configuration of the technology object.

6.2.3 Defining an interpreter program as preferred interpreter program of an interpreter (S7-1500T)

You can access an interpreter program technology object defined as a preferred interpreter program in the project navigation from the interpreter technology object.

Requirements

- An interpreter program technology object has been created.
- An Interpreter technology object has been created.

Procedure

To assign an interpreter program technology object to an interpreter technology object as a preferred interpreter program, follow these steps:

1. In the project navigation, drag-and-drop the interpreter program technology object onto the interpreter technology object.

Result

In the project navigation, a link to the interpreter program is created in the technology object under "Preferred interpreter programs".

When you open the programming editor using the link, the Interpreter technology object is already selected in the "Select Interpreter" drop-down list. The 3D visualization displays the kinematics that has been assigned to the Interpreter.

6.3 Data exchange between user program and interpreter program (S7-1500T)

To exchange data between the interpreter program and the user program of the CPU, you have the following two options:

- **Data exchange via the clipboard of the Interpreter technology object**

  The clipboard is a reserved area in the interpreter technology object.

  – <TO>.Clipboard.CbBool (Array [1..300] of Bool)
  – <TO>.Clipboard.CbDint (Array [1..100] of DInt)
  – <TO>.Clipboard.CbLreal (Array[1..100] of LReal)

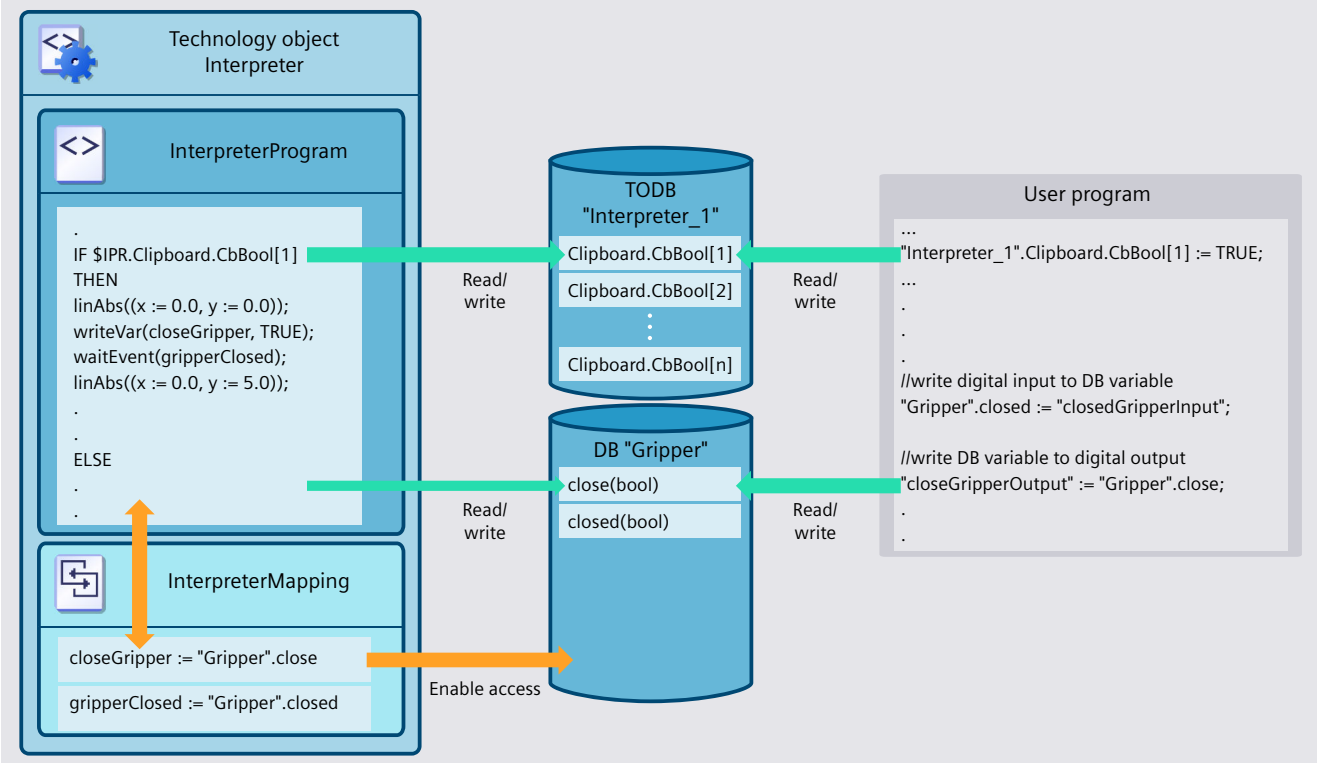
  You can read and write to the tags in the clipboard in both the interpreter program and the user program.

- **Data exchange via mapped tags in global data blocks**

  In Interpreter mapping, you can map global data block tags. The tags can have the data types BOOL, DINT, UDINT, DWORD , and LREAL.

  You can read and write to the mapped tags of data blocks both in the interpreter program and in the user program.

The following figure shows an example of the data exchange between interpreter program and user program when closing a gripper. The "writeVar()" instruction is used to write the "closeGripper" tag at the time of execution.



Comparison

	Data exchange via the clipboard of the interpreter	Data exchange via mapped tags of data blocks
Mapping required	-	✓
Symbolic program-ming	-	✓
Retentive tags are possible	-	✓
Data types	BOOL DINT LREAL	BOOL DINT UDINT LREAL DWORD
Number of tags	Fixed number 300 BOOL tags 100 DINT tags 100 LREAL tags	Number can be adjusted The Interpreter mapping technology object supports a maximum of 100 entries in the mapping table[1]
Access from multiple Interpreter programs	-	✓

[1] A table entry can contain an individual tag or an array.

Validity of the value when reading one tag in the interpreter program

The read access to a tag in the interpreter program provides the value valid at the time of program preparation.

How to access a value valid at the time of program execution is described in section Interpreter job sequence (Page 46).

Time of value change when writing one tag in the interpreter program

- Value assignment (e.g. closeGripper := TRUE;): The value is assigned for program preparation.
- Writing a tag with the "writeVar" instruction (for example, writeVar(closeGripper, TRUE);): The value is written to execute the job.

6.3.1 Accessing clipboard tags (S7-1500T)

Accessing clipboard tags in the interpreter program

In the interpreter program, you can access a clipboard tag as follows:

- Data type Bool
  ```
  $IPR.Clipboard.CbBool[i] // i = 1..300
  ```
- Data type Dint
  ```
  $IPR.Clipboard.CbDint[j] // j = 1..100
  ```
- Data type Lreal
  ```
  $IPR.Clipboard.CbLreal[k] // k = 1..100
  ```

Example:

```
...
IF $IPR.Clipboard.CbBool[1] THEN
   ptpAbs(pos3,trans := 0);
END_IF;
...
```

Accessing clipboard tags in the user program

To access a clipboard tag in the user program, specify the data area of the tag type with the correct index.

Example:

```
...
"Interpreter_1".Clipboard.CbBool[1] := TRUE;
...
```

6.3.2 Configuring start values for clipboard tags (S7-1500T)

Procedure

To update the start values for the clipboard tags, follow these steps:

1. In the configuration of the interpreter technology object, navigate to "Extended parameters > Clipboard".
2. Configure the start values for the clipboard tags:
   - BOOL tags [1-300]: In the "Start value project" column, select the start values for the BOOL tags from the drop-down list.
   - DINT tags [1-100]: In the "Start value project" column, enter the start values for the DINT tags.
   - LREAL tags [1-100]: In the "Start value project" column, enter the start values for the LREAL tags.

6.3.3 Mapping of technology objects (S7-1500T)

Requirements

- Configuration of the Interpreter mapping technology object is open.
- In the S7-1500 CPU, a speed axis, positioning axis or synchronous axis technology object has been added.

Procedure

To map a technology object in the interpreter mapping technology object, follow these steps:

1. In the configuration, navigate from the interpreter mapping technology object to "Mapping > Mapping of technology objects".
2. In the "Name in interpreter program" column, click the "Add" cell.
3. Enter the name under which the technology object is addressed in the Interpreter program.
4. In the "Technology object" column, select the technology object you want to map.

Result

The technology object is mapped.

6.3.4 Mapping of variables (S7-1500T)

In the Interpreter mapping technology object, you specify an assignment of PLC tags from global data blocks, technology object data blocks or instance data blocks to suitable variables in an Interpreter program.

Interpreter mapping supports the following data types for mapping tags:

- BOOL
- DINT
- UDINT
- DWORD
- LREAL
- Complex data types for describing position data or coordinate systems (e.g. TO_Struct_Ipr_Frame)
- ARRAYs of the data types named above

The Interpreter mapping technology object supports a total of a maximum of 100 entries in the mapping table. An entry can contain an individual tag or an array.

Requirement

- Configuration of the Interpreter mapping technology object is open.
- At least one supported data block has been created.
- The tag to be mapped is already defined.

Procedure

To map a tag from a data block in the Interpreter mapping technology object, follow these steps:

1. In the configuration, navigate from the Interpreter mapping technology object to "Mapping > Mapping of variables".
2. In the "Name in Interpreter program" column, click the "Add" cell.
3. Enter the name with which the variable is addressed in the Interpreter program.
4. In the "Data block variable" column, select the variable you want to map. The suitable data type is automatically selected.
5. To allow read-only access to the variable in the Interpreter program, select the check box in the "Read only" column.

NOTE

If the mapped variable in the data block has the property "Read only", then you must select the "Read only" check box for this variable in the mapping.

Result: The variable is mapped.

Mapping of complex data types

The following table shows which complex data types are supported in the Interpreter mapping and the corresponding, suitable data types of the mapped MCL tags.

PLC tag data type	MCL tag mapped data type
TO_Struct_Ipr_Frame	TO_Struct_Ipr_Frame
TO_Struct_Kinematics_Frame	
TO_Struct_Kinematics_KinematicsFrame	
TO_Struct_Ipr_Position	TO_Struct_Ipr_Position
TO_Struct_Ipr_AxPosition	TO_Struct_Ipr_AxPosition
TO_Struct_Ipr_JtPosition	TO_Struct_Ipr_JtPosition

User-defined data types of an identical structure are not supported in the Interpreter mapping.

Mapping of arrays

The Interpreter mapping supports mapping of arrays of supported data types. You can either map the complete array or a part of the array.

To map an array from a data block in the Interpreter mapping technology object, follow these steps:

1. In the configuration, navigate from the Interpreter mapping technology object to "Mapping > Mapping of variables".
2. In the "Name in Interpreter program" column, click the "Add" cell.
3. Enter the name with which the array is addressed in the Interpreter program.
4. In the "Data block variable" column, select the array you want to map. The suitable data type is automatically selected.
5. To map only a part of the array, adapt the table entry as follows.
   - In the "Start value index DB" line, enter the value as of which you want to map the part of the array element.
   - Adapt the array limits in the "Data type" line.

   Example: You map the tag "<TO_Kinematics>.OcsFrame" of "Array[1..3] of TO_Struct_KinematicsFrame" data type. To only map elements 2 and 3, enter the value 2 in the "Start value index DB" line and change the array limits to "Array[1..2] of TO_Struct_Ipr_Frame" in the "Data type" line.

Result: The array is mapped.

6.4 Preparing and executing an Interpreter program (S7-1500T)

Program preparation

The Interpreter technology object interprets the jobs of the loaded Interpreter program, prepares them in the MC_LookAhead acyclically, and includes the jobs in the Interpreter job sequence if required.

For program preparation, the Interpreter performs the following jobs:

- Job planning with the parameters valid for preparation
- Setting tags with the assignment operator ":="
- Planning the execution sequence by evaluating the control instructions, e.g. IF instructions or loops.

Program preparation has the advantage that kinematics motions including dynamics planning and program sequences can be calculated with sufficient lead time before they are executed. A "MC_LoadProgram" job in the user program starts the first preparation.

Interruption of program preparation

The Interpreter interrupts program preparation in the following situations:

- The maximum number of jobs to be prepared in the Interpreter job sequence (<TO_Interpreter>.Parameter.MaxNumberOfCommands) has been reached. As soon as a job has been executed, program preparation is continued.
- The maximum number of kinematics jobs (30) has been reached and another kinematics job is to be prepared. As soon as a kinematics job has been executed, program preparation is continued.

If you have to wait for data to be updated during program execution, you can use the following instructions in the Interpreter program to interrupt the program preparation:

- preHalt()
- waitEvent() with "mode" = 0
- powerOn()
- powerOff()
- defTool()
- setTool()

If program preparation is interrupted, the kinematics stops at the target position of the last motion job. No blending is therefore possible.

Examples for updating data

- Target position specification for motions: If the target position for a job is only available during program execution, you interrupt program preparation before the job with a "preHalt()" job.
- Control instructions (e.g. IF or CASE): If the expression is not to be evaluated until the time of execution, interrupt the program preparation before the control instruction with a "preHalt()" job, for example. The expression in the control instruction is evaluated with the current values at the time of program execution.

Program execution

When an Interpreter program is executed, the jobs from the job sequence are prepared in the MC_LookAhead organization block and then processed in the MC_Interpolator organization block.

An "MC_RunProgram" job in the user program triggers the execution of the Interpreter program. The Interpreter transfers the processed motion jobs to the technology objects controlled by the Interpreter. These technology objects execute the transferred jobs cyclically in the MC_Interpolator. During execution of the Interpreter program, the Interpreter continuously prepares the next jobs.

As soon as the Interpreter has finished executing the Interpreter program, the Interpreter program is prepared for another execution. The preparation of the Interpreter program starts again. Tags are initialized again using ":=" and jobs are prepared. The Interpreter program can then be executed again.

Overview of the time when instructions were executed

The following instructions are executed with the program preparation:

- setAxisDyn
- setAxisDynMax
- setDyn
- setDynMax
- setPlane
- setCircDirs
- setDynAdapt
- setOriDyn
- setOriDynMax
- setPtpDyn
- setCs
- setBlendDist
- setBlendFactor
- setBlend
- setTrans
- setLc
- setTurnJoint
- setOvr
- preHalt

The following instructions are executed when the program is executed:

- writeVar
- waitEvent
- waitTime

- setControlledByInterpreter
- powerOn
- powerOff
- home
- move
- posRel
- posAbs
- torqueLimitOn
- torqueLimitOff
- linAbs
- linRel
- circAbs
- circRel
- ptpAbs
- ptpRel
- ptpAxAbs
- ptpAxRel
- ptpJtAbs
- ptpJtRel
- defOcs
- defTool
- setTool
- trackIn
- defWsZone
- defKinZone
- setWsZoneActive
- setWsZoneInactive
- setKinZoneActive
- setKinZoneInactive

See also

Examples for preparation and execution of an Interpreter program (Page 48)

6.4.1 Interpreter job sequence (S7-1500T)

The length of the Interpreter job sequence specifies the maximum number of jobs that can be prepared by the Interpreter. The Interpreter job sequence includes all jobs of the Interpreter program. You can set the maximum number of jobs to be prepared between 10 and 100 (<TO_Interpreter>.Parameter.MaxNumberOfCommands).

The Interpreter job sequence can contain up to 30 kinematic motion jobs. This limitation is fixed irrespective of the value of the kinematics job sequence configured at the kinematics technology object (<TO_Kinematics>.MotionQueue.MaxNumberOfCommands).

When loading the Interpreter program and during program execution, the Interpreter prepares the jobs until the maximum number of jobs is reached. During execution of the Interpreter program, the jobs or motion sequences are executed as soon as they are prepared and the previous job is completed.

The Interpreter calculates the jobs in advance during program preparation. The job is executed with the parameters at the time of preparation and not with the parameters at the time of execution.

Configuring the Interpreter job sequence

To configure the job sequence of the Interpreter, follow these steps:

1. Open the "Extended parameters > Interpreter job sequence" configuration window of the Interpreter technology object.
2. Enter the desired value in the "Maximum number of jobs to be prepared" field (<TO_Interpreter>.Parameter.MaxNumberOfCommands).

6.4.2 Maximum wait time (S7-1500T)

With the maximum wait time you can determine the maximum time between the possible and the effective start of an job. As soon as the previous job has been processed and the next job has been prepared, it is possible to start the new job.

In a motion sequence with multiple jobs, the interpreter technology object attempts to prepare all jobs belonging to this motion sequence in advance before starting. If you set the maximum wait time so that the interpreter can only process a subset of the jobs within this time, the first job of the motion sequence starts before the motion sequence is fully prepared. Further processing runs parallel to the execution of the motion sequence.

For the maximum wait time, you can set values between 0.0 s and 2.0 s. If you define the maximum wait time as 0.0 s, the execution of the motion sequence starts as soon as the motion sequence is fully prepared.

Example without a fully populated interpreter job sequence

The preparation of a motion sequence takes 1.5 s.

With a maximum wait time of 0.0 s, the execution of the motion sequence starts after 1.5 s, as soon as the motion sequence is fully prepared.

With a maximum wait time of 1.0 s, the execution of the motion sequence starts 1.0 s after the first job has been prepared. The execution of the motion sequence starts regardless of the fact that the motion sequence is not yet fully prepared at this time.

With a maximum wait time of 2.0 s, the execution of the motion sequence starts after 1.5 s, as soon as the motion sequence is fully prepared.

Configuring the maximum wait time

To configure the maximum wait time, follow these steps:

1. Open the configuration window "Extended parameters > Program preparation" of the interpreter technology object.
2. In the "Maximum wait time" field, enter the desired value (<TO_Interpreter>.Parameter.StartTimeout).

6.4.3 Program override (S7-1500T)

The program override allows you to set a factor that affects the velocity and acceleration or deceleration of programmed axis movements and kinematics motions. The program override is taken into account by the Interpreter technology object when processing the motions. Changes to the program override do not affect motion jobs that have already been prepared. You can set values between 1% and 100%.

The program override works in addition to the override values of the technology objects to be controlled. For example, if the program override is set to 50% and the velocity override of a Kinematics technology object is also 50%, the resulting override value is 25%, which affects the kinematic motion speeds. The resulting override value for acceleration and deceleration is 50%. Any changes in the override values of the technology objects to be controlled have a direct influence on active jobs.

The start value of the program override can be defined in the configuration of the Interpreter technology object. The start value takes effect modally when loading the Interpreter program and can be changed via the MCL instruction "setOvr()".

Configuring the start value of the program override

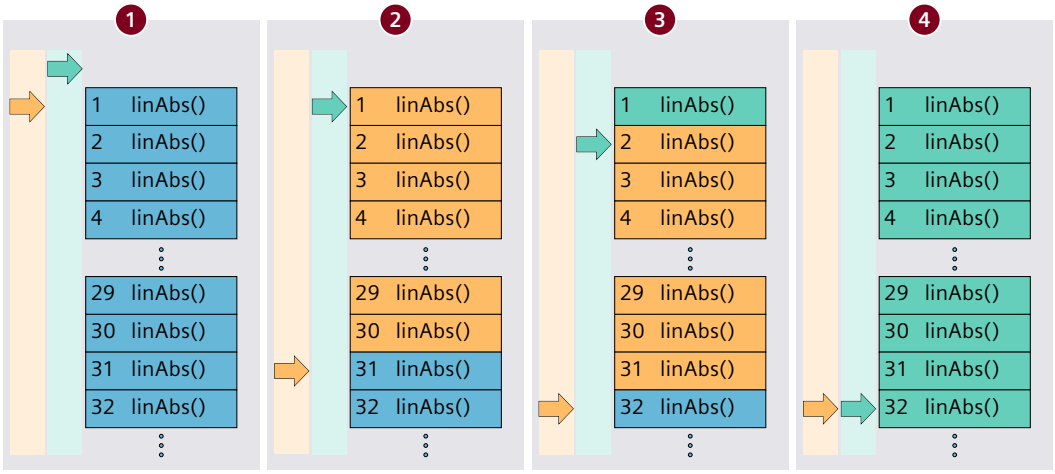
To configure the start value of the program override, follow these steps:

1. Open the configuration window "Extended parameters > Program preparation" of the Interpreter technology object.
2. In the "Start value program override" field, enter the desired value (<TO_Interpreter>.Parameter.ProgramOverride).

6.4.4 Examples for preparation and execution of an Interpreter program (S7-1500T)

Maximum number of kinematics jobs prepared

This example describes how program preparation and execution behave when the maximum number of kinematics jobs is reached.



Preparation active

Execution active

Job to be prepared

Prepared job

Executed job

① As soon as the Interpreter program is loaded into the Interpreter with the "MC_LoadProgram" instruction, the Interpreter begins to prepare the kinematics jobs.

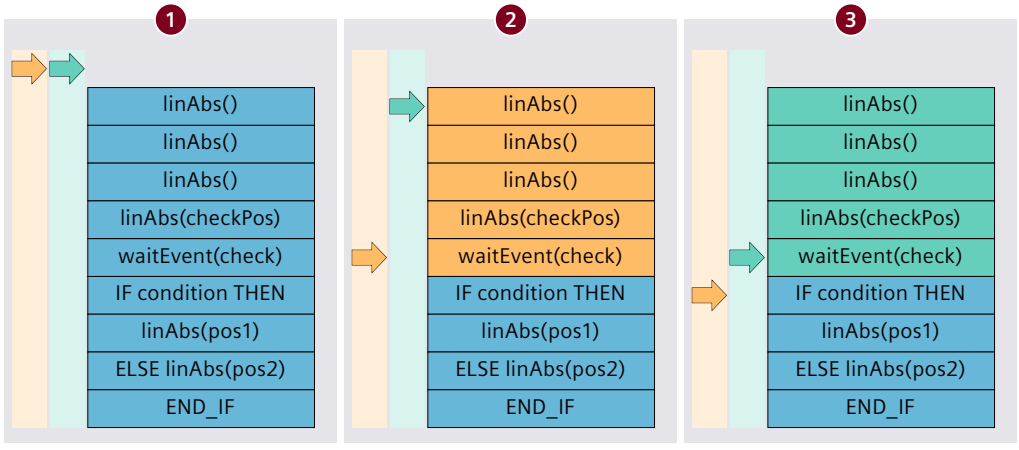
② When the Interpreter has prepared the 30th kinematics job, the maximum number of the Interpreter job sequence for kinematics jobs has been reached. The Interpreter interrupts the program preparation.
With the "MC_RunProgram" instruction, the Interpreter starts executing the kinematics jobs.

③ The Interpreter is busy with program preparation and program execution at the same time. At no time is program preparation more than 30 kinematics jobs ahead of program execution. Program execution can progress faster than the program preparation.

④ Once the last kinematics job is processed, the Interpreter executes the outstanding kinematics jobs up to the last job.

Evaluating an expression during program execution

This example describes how the program preparation and execution behave for a job of the MCL instruction "waitEvent()".



Preparation active

Execution active

Job to be prepared

Prepared job

Executed job

① In the Interpreter program, an evaluation is performed at the "checkPos" position to determine which of the positions, "pos1" or "pos2", is to be approached. This involves approaching the "checkPos" position and waiting for the "check" evaluation.
The "check" waiting condition is specified e.g. via a mapping variable.
The mapping variable is assigned a digital input in the user program.

② Program preparation is interrupted with the "waitEvent(check)" job. The program execution continues after this.

③ The program execution also reaches the "waitEvent(check)" job. Program execution is interrupted until the mapping variable returns "check" = TRUE.

Reading and writing variables

This example shows the behavior of read and write accesses in the Interpreter program.

An assignment with ":=" reads the value and writes the tag for program preparation.

A "writeVar(variable, value)" job writes the variable for program execution. The value is read during program preparation.

MCL

```
PROGRAM main
  VAR
  END_VAR
  $IPR.Clipboard.CbDint[1] := 5; // set to 5 during preprocessing
  $IPR.Clipboard.CbDint[2] := 5; // set to 5 during preprocessing
  $IPR.Clipboard.CbDint[100] := 3; // set to 3 during preprocessing
  writeVar( $IPR.Clipboard.CbDint[100], 10 ); // set from 3 to 10 during execution
  $IPR.Clipboard.CbDint[1] := $IPR.Clipboard.CbDint[100]; // set to var1 (3) during prepro-
cessing
```

MCL

```
  (* set to $IPR.Clipboard.CbDint[100] (3) during execution, because 3 was the $IPR.Clip-
board.CbDint[100] value during preprocessing,
     can be 10 in case of slow preprocessing - if instruction above is already executed
     and this instruction is preprocessed afterwards *)
  writeVar( $IPR.Clipboard.CbDint[1], $IPR.Clipboard.CbDint[100] );
  preHalt( ); // use preHalt to interrupt preprocessing
  (* set to $IPR.Clipboard.CbDint[100] (10) during execution, because preprocessing contin-
ues after preHalt() and
     new value setted during execution of $IPR.Clipboard.CbDint[100] is used for prepro-
cessing, can be 10 in case
     of slow preprocessing - if instruction above is already executed and this instruction
     is preprocessed afterwards *)
  writeVar( $IPR.Clipboard.cbDint[2], $IPR.Clipboard.CbDint[100] );
END_PROGRAM
```

6.4.5 Variables: Preparation of the Interpreter program (S7-1500T)

The following variables of the Interpreter technology object are relevant for the preparation of the Interpreter program:

Variable	Description	
<TO>.Parameter.MaxNumberOfCommands	Maximum number of jobs to be prepared in the Interpreter job sequence	
<TO>.Parameter.StartTimeout	Maximum wait time Maximum time between the possible and effective start of an order in [s] An incompletely prepared motion sequence will start after the expiry of the maximum wait time.	
	0.0	Motion sequences only start when they are fully prepared or the maximum number of jobs to be prepared in the Interpreter job sequence has been reached.
<TO>.Parameter.ProgramOverride	Start value of the program override in [%] The percentage factor acts as the start value of the modal parameter when preparing the dynamic profiles of axis and kinematics motions.	
<TO>.StatusInterpreter.LevelOfPreparedCommands	Utilization of the program preparation in [%] in 5% steps: Display of the maximum value of the following criteria: • Prepared Interpreter jobs • Processed kinematics jobs • Prepared synchronous actions	

6.5 Executing the interpreter program in the interpreter technology object (S7-1500T)

6.5.1 Loading/unloading the Interpreter program (S7-1500T)

To run an Interpreter program, first load the Interpreter program into the Interpreter technology object. The Interpreter program is prepared and is then ready for execution.

After changing the operating state to STOP or POWER OFF of the CPU, you must reload the Interpreter program into the Interpreter technology object.

To load a modified or different Interpreter program into the Interpreter technology object, first unload the loaded Interpreter program from the Interpreter technology object. You can then load the modified or new Interpreter program into the Interpreter technology object.

Parameter inputs

Use the following parameters of the "MC_LoadProgram (Page 343)" Motion Control instruction to determine the loading behavior of the Interpreter technology object:

- With the parameter "Mode" = 1 you can define the loading mode "Load".

  "Downloading" in download mode:

  - Use the "Program" parameter to define the name of the Interpreter program technology object as a string, for example, 'InterpreterProgram_1'.
  - Use the "ProgramSource" parameter to define the source of the Interpreter program.
  - Use the "Mapping" parameter to optionally define the name of the Interpreter mapping technology object as a string, for example, 'InterpreterMapping_1'.
  - With the parameter "MappingSource" = 1 you can define the Interpreter mapping technology object defined by the "Mapping" parameter as the source of the Interpreter mapping.

- With the parameter "Mode" = 0 you can define the loading mode "Unload".

Source of the Interpreter program

With "ProgramSource" = 1, the program is loaded from the Interpreter program technology object defined by the "Program" parameter into the Interpreter technology object.

Loading the Interpreter program into the Interpreter technology object.

After the start of the "MC_LoadProgram" job with "Mode" = 1 and "Execute" = TRUE, the Interpreter program is prepared for execution, together with any specified mapping in the Interpreter technology object. As soon as the job is completed with "Done" = TRUE, the Interpreter program is ready to run.

Unloading the Interpreter program from the Interpreter technology object

After the start of the "MC_LoadProgram" job with "Mode" = 0 and "Execute" = TRUE, the Interpreter program is unloaded from the Interpreter technology object, together with any specified mapping. The Interpreter technology object is then ready to load an Interpreter program again.

6.5.2 Starting execution of the interpreter program (S7-1500T)

The "MC_RunProgram (Page 345)" Motion Control instruction starts the execution of an interpreter program loaded in the interpreter technology object.

Start program execution

With "Execute" = TRUE, you can start the "MC_RunProgram" job. Even if the loading process is not yet complete ("<TO>.StatusWord.X9" = TRUE (Loading)), you can already issue an "MC_RunProgram" job. The execution of the interpreter program starts as soon as the interpreter program is prepared and loaded ("<TO>.StatusWord.X10" = TRUE (Loaded)).

During program execution

As long as the interpreter technology object is executing the interpreter program, the "MC_RunProgram" job is in progress and "Active" = TRUE ("<TO>.StatusWord.X5" = TRUE (InRun)).

After program execution

As soon as the interpreter technology object has finished executing the interpreter program, the interpreter program is prepared to carry out another execution. The preparation of the interpreter program starts again. All parameters are reinitialized. Processing of the "MC_RunProgram" job is then completed with "Done" = TRUE.

Now you can restart execution of the interpreter program.

6.5.3 Stopping execution of the interpreter program (S7-1500T)

You can use the "MC_StopProgram" Motion Control instruction to stop the execution of the Interpreter program in the interpreter technology object. A single axis/kinematics controlled by the interpreter is stopped ("<TO>.StatusInterpreterMotion.StatusWord.X0" = TRUE (ControlledByInterpreter)).

NOTE

Never-ending motion jobs

Note that for motion jobs that do not end automatically, e.g. "move()", the "<TO>.StatusInterpreterMotion.StatusWord.X0" tag (ControlledByInterpreter) of the corresponding technology object is set to "FALSE" as soon as the technology object has reached the specified state, e.g. the specified velocity.

To enable termination of these motion jobs with an "MC_StopProgram" job, use the MCL instruction "setControlledByInterpreter()" and set the "<TO>.StatusInterpreterMotion.StatusWord.X0" tag (ControlledByInterpreter) back to "TRUE".

Parameter inputs

You can determine the dynamic behavior with the following parameters of the "MC_StopProgram (Page 347)" Motion Control instruction:

- With the "Mode" parameter, you can define the dynamic behavior of the single axis/kinematics for the stop motion.

Mode for dynamic behavior

With "Mode" = 0, the single axis/kinematics stops with maximum dynamics. The current motion job is terminated.

With "Mode" = 1, the single axis/kinematics stops with the dynamics specified for the current motion job. The current motion job is terminated.

With "Mode" = 2, the single axis/kinematics executes the current motion job or motion sequence. A motion sequence refers to several motion jobs that are blended into one another. The single axis/kinematics then stops.

Stop program execution

With "Execute" = TRUE you can stop the execution of the interpreter program. The motion of the single axis/kinematics controlled by the interpreter technology object is stopped depending on the specified mode. The "MC_StopProgram" job is processed until the stop motion is complete.

As long as "Execute" = TRUE, additional jobs for the interpreter technology object are rejected ("<TO>.StatusWord.X7" = TRUE (Stopping)).

After stopping program execution

As soon as the single axis/kinematics have come to a standstill, the "MC_StopProgram" job is completed ("Done" = TRUE). The corresponding technology object of the single axis/kinematics is then no longer controlled by the interpreter technology object ("<TO>.StatusInterpreterMotion.StatusWord.X0" = FALSE (ControlledByInterpreter)).

To then release the interpreter technology object for new jobs, reset the "Execute" parameter of the "MC_StopProgram" job to "FALSE".

6.5.4 Modifying and loading the Interpreter program (S7-1500T)

To run a modified interpreter program, you first need to unload the loaded Interpreter program from the Interpreter technology object, and then load the modified Interpreter program into the Interpreter technology object. You can then start the execution of the Interpreter program.

Requirements

- You have loaded the Interpreter program in the Interpreter technology object
- You have changed the loaded Interpreter program.

Procedure

To load the changes of an Interpreter program, follow these steps:

1. If the loaded interpreter program is running, stop it from executing with "MC_StopProgram.Execute" = TRUE.

   The motion of a single-axis/kinematics controlled by the Interpreter technology object is stopped, depending on the specified mode. As soon as the job is completed with "Done" = TRUE, the single axis/kinematics comes to a standstill and the execution of the Interpreter program is stopped.

2. Unload the Interpreter program from the Interpreter technology object with "MC_LoadProgram.Mode" = 0 and "Execute" = TRUE.

   As soon as the job is completed with "Done" = TRUE, the Interpreter technology object is ready to load an Interpreter program.

3. Load the Interpreter program technology object to the CPU.

4. Load the changed Interpreter program with "MC_LoadProgram.Mode" = 1 and "Execute" = TRUE to the Interpreter technology object.

   As soon as the job is completed with "Done" = TRUE, the Interpreter program is ready to run.

Result

The modified Interpreter program is loaded in the Interpreter technology object and ready for execution. Start the execution of the Interpreter program with "MC_RunProgram.Execute" = TRUE.

6.5.5 Variables: Execution of the Interpreter program (S7-1500T)

Interpreter technology object

The following variables of the Interpreter technology object are relevant for executing the Interpreter program:

Variable	Description	
<TO>.ProgramName	Name of the currently loaded technology object Interpreter program	
<TO>.ProgramSource	Source of the currently loaded Interpreter program	
	0	No Interpreter program loaded
	1	Interpreter program technology object
<TO>.MappingName	Name of the currently loaded Interpreter mapping technology object	
<TO>.MappingSource	Source of the currently loaded Interpreter mapping	
	0	No Interpreter mapping loaded
	1	Interpreter mapping technology object
<TO>.ActualLineNumber	Line number of the currently executed program line or the last executed program line	
<TO>.StatusWord.X0 (Control)	No Motion Control job is active at the Interpreter technology object.	
<TO>.StatusWord.X5 (InRun)	The technology object is executing an Interpreter program.	
<TO>.StatusWord.X6 (Done)	Execution of the Interpreter program has finished.	
<TO>.StatusWord.X7 (Stopping)	Execution of the Interpreter program will be or has been stopped.	
<TO>.StatusWord.X9 (Loading)	The technology object is loading the Interpreter program. Interpreter program preparation is running.	
<TO>.StatusWord.X10 (Loaded)	The Interpreter program is loaded and prepared.	

Technology objects controlled by the Interpreter

The following variables of the technology object controlled by the Interpreter are relevant when executing the Interpreter program:

Variable	Description
<TO>.StatusInterpreterMotion.Interpreter	For a technology object of an axis: Controlling Interpreter technology object
<TO>.StatusInterpreterMotion.StatusWord.X0 (ControlledByInterpreter)	Is set to the value "TRUE" when an MCL job is prepared or active, or the bit is set via the MCL instruction "setControlledByInterpreter()".
<TO>.StatusInterpreterMotion.StatusWord.X1 (MotionByInterpreter)	Is set to the value "TRUE" when an MCL motion job is in effect.

6.6 Testing Interpreter program (S7-1500T)

With technology version 8.0, the Interpreter only supports the "Automatic" program mode.

As of technology version 9.0, the Interpreter supports the "Automatic" and "Debug" program modes.

Functions in "Debug" program mode

The "Debug" program mode offers the following functions:

- Controlling the Interpreter program via the toolbar of the programming editor (Page 65):
  - Interrupt program
  - Resume program
  - Stop program
  - Run program step by step
- Display tag information in "Debug" program mode (Page 68)
- Defining breakpoints for debugging (Page 63)

Master control in "Debug" program mode

In "Debug" program mode, the program execution by the Interpreter can be influenced via the user program of the CPU and the toolbar in the programming editor.

6.6.1 Guideline of testing of the Interpreter program (S7-1500T)

The consideration of the breakpoints depends on whether the Interpreter program is prepared and executed for the first time or has already been prepared.

Requirements

- In the programming editor, you have selected the Interpreter technology object with which you want to test the Interpreter program.
- You have selected the Interpreter mapping technology object in the programming editor with which you want to test the Interpreter program.

Test an Interpreter program prepared and executed for the first time

To test an Interpreter program for the first time, proceed as follows.

1. Loading (Page 51) the Interpreter program into the Interpreter technology object.
2. Activate (Page 58) the "Debug" program mode.
3. Set and activate (Page 63) the required breakpoints.
4. Start (Page 52) the execution of the Interpreter program.
5. Control (Page 65) the Interpreter program step by step using the toolbar in the programming editor.

   All activated breakpoints are taken into account.

Test an already prepared Interpreter program

To test an Interpreter program that has already been loaded and prepared, proceed as follows.

1. Activate (Page 58) the "Debug" program mode.
2. Set and activate (Page 63) the required breakpoints.

   The breakpoints are taken into account during the next program preparation.

   The program execution is interrupted at the first prepared breakpoint.
3. Control (Page 65) the Interpreter program step by step using the toolbar in the programming editor.

6.6.2 Activating "Debug" program mode (S7-1500T)

⚠ WARNING

Unexpected axis motions

During operation in "Debug" program mode, the kinematics can execute unexpected motions, for example, due to incorrect configuration of the drive or technology object. The user program continues to run parallel to the Interpreter program in the "Debug" program mode. The user program can be used to move additional axes, including synchronized following axes.

Possible effects:

- Human life or health may be endangered.
- The machine or the entire plant may be damaged.

Therefore, carry out the following protective measures before activating the "Debug" program mode:

- Ensure that the EMERGENCY OFF switch is within the reach of the operator.
- Enable the hardware limit switches.
- Enable the software limit switches.
- Use blocked zones to prevent mechanical collisions.
- Ensure that following error monitoring is enabled and correctly set.

⚠ WARNING

Changed behavior of the user program in "Debug" program mode

The motion control instructions "MC_StopProgram" and "MC_Reset" with "Restart" = TRUE are **not** executed for the Interpreter technology object in "Debug" program mode.

Possible effects:

- Human life or health may be endangered.
- The machine or the entire plant may be damaged.

Carry out the following protective measures:

- Ensure that the EMERGENCY OFF switch is within the reach of the operator.
- Make sure that, despite interrupting the Interpreter program, the user program runs correctly and securely.
- Note the various options for resuming the Interpreter program.

⚠ WARNING

Changed behavior of the Interpreter program in "Debug" program mode

The Interpreter program is executed with a delay. The Interpreter program works with the values from the time of preparation. These values may have already been changed by the user program when the job is done in the Interpreter program.

The dynamics and path of kinematics can differ between the "Debug" and "Automatic" program modes.

Possible effects:

- Human life or health may be endangered.
- The machine or the entire plant may be damaged.

Carry out the following protective measures:

- Ensure that the EMERGENCY OFF switch is within the reach of the operator.
- Check which values the Interpreter program works with.
- Select the appropriate option to resume the Interpreter program.

⚠ WARNING

Stop "Debug" program mode

You set the conditions for termination as soon as you activate the "Debug" program mode. Program execution may resume unexpectedly when switching to the "Automatic" program mode.

Possible effects:

- Human life or health may be endangered.
- The machine or the entire plant may be damaged.

Carry out the following protective measures:

- Make sure that you set the appropriate behavior when activating the "Debug" program mode.

Requirements

- A project with a configured Interpreter technology object has been downloaded to the CPU.
- The CPU is in "RUN" mode.
- No control panel has control priority over the connected kinematics and axes.
- No "MC_StopProgram" job is active on the Interpreter technology object.
- No "MC_Reset" job with "Restart" = TRUE is active on the Interpreter technology object.
- In the programming editor, the Interpreter technology object is selected.

Procedure

Proceed as follows to activate the "Debug" program mode:

1. Click on the "Activate "Debug" program mode" button in the programming editor.

   The "Activate "Debug" program mode" window opens.

2. In the "Override mode" drop-down list, select how the Interpreter behaves when the "Debug" program mode is exited:

   – Continue program execution

     The Interpreter switches to the "Automatic" program mode and automatically continues the program execution on the program line, at which the "Debug" program mode is finished. Program execution is also continued if program execution was interrupted in Debug mode.

   – Abort program execution

     If the execution of the Interpreter program is interrupted at the time of finishing, the interpreter switches to "Automatic" program mode and aborts the Interpreter program at the current program line. The program status remains at "Interrupted".

     When the Interpreter program is executed, the Interpreter switches to the "Automatic" program mode and automatically continues the program execution on the program line at which the "Debug" program mode is finished.

3. From the "Timeout Mode" drop-down list, select how the Interpreter should behave if the online connection is lost:

   – Stop "Debug" program mode

     The Interpreter switches to "Automatic" program mode. Active "MC_RunProgram" jobs are aborted with a technology alarm. Active motions are stopped with maximum dynamics.

   – Continue "Debug" program mode

     If the execution of the interpreter program is interrupted at the time of the timeout, the interpreter switches to "Automatic" program mode. The program status remains at "Interrupted".

     When the Interpreter program is executed, the active job or the active motion sequence is ended and the "Automatic" program mode is activated. Active "MC_RunProgram" jobs are aborted with a technology alarm. The program status is set to "Interrupted".

     You can re-activate the "Debug" program mode and continue running the Interpreter program on the current program line.

4. In the "Monitoring time" input field, configure the time after which the behavior configured in "Timeout mode" takes effect following a loss of the online connection.
5. Click "OK".

Result

The "Debug" program mode is activated (<TO>.StatusInterpreter.ProgramMode = 2).

The toolbar of the programming editor has the master control over the Interpreter technology object (<TO>.StatusWord.X4 = TRUE (MasterControlActive)).

Monitoring time

The "Debug" program mode requires a direct connection from its TIA Portal to the controller, which is monitored cyclically. If the Interpreter receives no sign-of-life from the PG/PC within the monitoring time, the Interpreter behaves in accordance with the configured timeout mode.

Monitoring time	Effect
Too low	The "Debug" program mode is often interrupted because the monitoring time is exceeded. The behavior configured in "Timeout mode" applies.
Appropriate	The monitoring time is not exceeded and the "Debug" program mode is not interrupted. Recommendation: 1000 ms to 2000 ms
Too high	The "Debug" program mode can only be activated again after the set monitoring time has elapsed. The Interpreter program continues to be executed from the loss of the online connection until the monitoring time has elapsed.

6.6.3 Behavior of the user program in "Debug" program mode (S7-1500T)

You can execute the following Motion Control instructions on the Interpreter:

Motion Control instruction	"Debug" program mode	"Automatic" program mode
MC_LoadProgram	✓	✓
MC_RunProgram	✓	✓
MC_StopProgram	-	✓
MC_Reset with "Restart" = FALSE	✓	✓
MC_Reset with "Restart" = TRUE	-	✓

6.6.4 Behavior of the Interpreter program in "Debug" program mode (S7-1500T)

Preparation of the Interpreter program

In "Debug" program mode, the Interpreter technology object prepares the Interpreter program in the same way as in "Automatic" program mode.

If you control the Interpreter program via the toolbar of the programming editor, you can choose whether or not to restart the program preparation before the next program step.

Response to blending

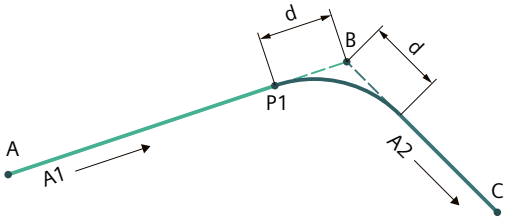
As long as a motion sequence with blending is executed without breakpoints, the motions are blended in the same way as in "Automatic" program mode.

When a motion sequence with blending is interrupted by a breakpoint after a linear or circular motion, the kinematics stops at the following position P1:

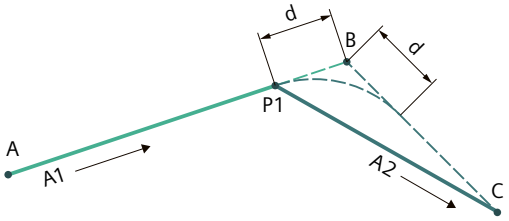
P1 = target position B — blending distance d

The path up to this position P1 corresponds to the path in the "Automatic" program mode.

If you resume the Interpreter program and use the existing program preparation, the path corresponds to the programmed blending behavior.



If you resume the Interpreter program and restart the program preparation, the programmed blending behavior is not taken into account. The kinematics move directly from position P1 to target position C.



When a motion sequence is interrupted with an SPTP motion, the kinematics stop at the target position of the job with the breakpoint. The blending is ignored.

Response to SYNC instructions

Synchronized actions are treated as an instruction.

If you insert a breakpoint before the SYNC instruction, the Interpreter program stops at breakpoint. If you execute another step of the Interpreter program, the entire SYNC instruction is executed. A breakpoint within the SYNC instruction is not taken into account.

Example

MCL

```
linAbs( myPos1, trans := 1, blend := 2 ); // for assigned TO_Kinematics
SYNC
  linAbs( myPos2, trans := 1, blend := 2 ); // for assigned TO_Kinematics
  linAbs( myPos3 ); //for assigned TO_Kinematics
ON_START
  posAbs( myAxis1, 5.0 );
  posAbs( myAxis2, 10.0 );
END_SYNC;
posRel( myAxis1, 0.0 );
```

6.6.5 Defining breakpoints for debugging (S7-1500T)

You can test an Interpreter program by having each MCL instruction executed individually, step by step. To do this, you set breakpoints into the program code, establish an online connection and enable the breakpoints in the Interpreter program. You manage the breakpoints in the programming editor.

In "Debug" program mode, the program execution is interrupted when a breakpoint is reached. The "<TO>.StatusWord.X8 (Interrupted)" tag is set to the "TRUE" value.

You can monitor and change the actual values of tags at a breakpoint. You can then resume the program execution via the toolbar of the programming editor.

Set/remove breakpoints

You can set a maximum of 10 breakpoints in an Interpreter program. You can set and remove breakpoints both offline and online, and in "Debug" program mode.

To set a breakpoint for a line in the Interpreter program, click on the left in the column before the line numbering in the programming editor. The breakpoint is indicated with an icon. If a breakpoint and a bookmark have been set in a line, the breakpoint icon is displayed before the bookmark icon.

To remove a breakpoint, click on the icon of an existing breakpoint. The symbol is not displayed anymore. To remove all breakpoints, select the "Remove all breakpoints" menu item from the context menu of a breakpoint.

Activate/deactivate breakpoint

Breakpoints can be activated or deactivated. You can activate and deactivate breakpoints both offline and online, and in "Debug" program mode. In the "Debug" program mode, only activated breakpoints are taken into account when preparing and executing the Interpreter program.

To activate a deactivated breakpoint, select the "Activate breakpoint" menu item from the context menu of a deactivated breakpoint. To activate all breakpoints, select the "Enable all breakpoints" context menu item.

To deactivate an activated breakpoint, select the "Deactivate breakpoint" menu item in the context menu of an activated breakpoint. To deactivate all breakpoints, select the "Disable all breakpoints" context menu item.

Status of breakpoints

The following table shows the possible status values of the breakpoints:

Icon	Description
	Deactivated breakpoint
	Activated breakpoint
	Prepared breakpoint (only in "Debug" program mode)
	The breakpoint could not yet be prepared (only in "Debug" program mode).
	An error occurred while preparing the breakpoint (only in "Debug" program mode).

6.6.6 Controlling the Interpreter program via the toolbar of the programming editor (S7-1500T)

In the "Debug" program mode, you can control the program execution via the toolbar of the programming editor. The buttons for controlling the Interpreter are displayed as soon as "Debug" program mode is activated.

You cannot start the program execution via the toolbar in the programming editor, but only with a "MC_RunProgram" job in the user program. As soon as the execution of the Interpreter program is started with a "MC_RunProgram" job, the buttons in the toolbar of the programming editor are active.

The line of program code that is currently executed is marked with the symbol.

Restart program preparation

For actions that continue the Interpreter program execution, you can choose whether the program preparation is to be restarted:

- No restart of program preparation

  The Interpreter continues using the existing program preparation. Values changed in the user program are not updated in the Interpreter program.

  Even with interruptions, the kinematics motion is identical to the motion in the "Automatic" program mode. The blending between motion jobs is taken into account.

  The buttons in the toolbar of the programming editor contain green icons, e.g. .

- Restart of program preparation

  The Interpreter prepares the Interpreter program for execution. Values changed by the user program are updated in the Interpreter program. Results from the Interpreter program are updated for the user program.

  The kinematics motion is different from the motion in the "Automatic" program mode. The blending between motion jobs is not taken into account.

  The buttons in the toolbar of the programming editor contain black icons, e.g. .

Interrupting execution of the Interpreter program

If the Interpreter program is running (<TO>.StatusWord.X5 (InRun)), you can interrupt the execution of the Interpreter program via the toolbar of the programming editor. The active motions on the connected technology objects are stopped.

You can select the timing of when the motion is stopped following the interruption:

- To stop at the end of the active Interpreter job, click on the button.
- To stop at the end of the active motion sequence, click on the button.

The tag "<TO>.StatusWord.X8 (Interrupted)" = TRUE indicates that the execution of the Interpreter program has been interrupted.

Execute the Interpreter program step by step

If the Interpreter program has been interrupted (<TO>.StatusWord.X8 (Interrupted)), you can execute the Interpreter program stepwise via the toolbar of the programming editor. If a breakpoint occurs during stepwise execution, the program execution is stopped a breakpoint.

The following options are available to you for stepwise execution:

Function	Button	Description
Step over		The Interpreter continues using the existing program preparation. The Interpreter executes the next MCL instruction in the main program/in a called function. The program execution is stopped within the main program/function before the MCL instruction after the next one. If the next instruction contains a function call, the complete function is executed and is stopped before the first MCL instruction after the function call.
		The Interpreter prepares the Interpreter program for execution. The Interpreter executes the next MCL instruction in the main program/in a called function. The program execution is stopped within the main program/function before the MCL instruction after the next one. If the next instruction contains a function call, the complete function is executed and is stopped before the first MCL instruction after the function call.
Step in		The Interpreter continues using the existing program preparation. The Interpreter executes the next MCL instruction in the main program/in a called function. The program execution is stopped within the main program/function before the MCL instruction after the next one. If the next MCL instruction contains a function call, the program execution is stopped before the first MCL instruction of the called function.
		The Interpreter prepares the Interpreter program for execution. The Interpreter executes the next MCL instruction in the main program/in a called function. The program execution is stopped within the main program/function before the MCL instruction after the next one. If the next MCL instruction contains a function call, the program execution is stopped before the first MCL instruction of the called function.
Step out		The Interpreter continues using the existing program preparation. Within the main program, the Interpreter executes the next MCL instruction. The program execution is stopped before the MCL instruction after the next one. The Interpreter executes the complete function within the called function. The program execution is stopped at the function call.

Function	Button	Description
Step out		The Interpreter prepares the Interpreter program for execution. Within the main program, the Interpreter executes the next MCL instruction. The program execution is stopped within the main program before the MCL instruction after the next one. The Interpreter executes the complete function within the called function. After the execution of the function, the program execution is stopped again at the function call.

Continuing execution of the Interpreter program

If the execution of the Interpreter program is interrupted (<TO>.StatusWord.X8 (Interrupted)), you can continue the execution of the Interpreter program via the toolbar of the programming editor.

When you resume the program execution, you can select whether to prepare the Interpreter program again:

- To use the existing program preparation, click the  button.
- To reprocess the Interpreter program, click on the  button.

The tag "<TO>.StatusWord.X5 (InRun)" = TRUE indicates that Interpreter program is running.

Canceling execution of the Interpreter program

You can cancel the execution of the Interpreter program via the toolbar of the programming editor. The active motions on the connected technology objects are stopped.

You can select the timing of when the motion is stopped following the interruption:

- To stop the active motion, click the  button.
- To stop at the end of the active Interpreter job, click on the  button.
- To stop at the end of the active motion sequence, click on the  button.

The tag "<TO>.StatusWord.X7 (Stopping)" = TRUE indicates that the execution of the Interpreter program has been canceled.

6.6.7 Displaying tag values and making changes in "Debug" program mode (S7-1500T)

In "Debug" program mode, you can display the value of a tag of the and change the value, e.g. for test purposes.

Requirements for displaying the value of a tag

- The Interpreter is in "Debug" program mode.
- The Interpreter program is in the "Interrupted" status, for example, at a breakpoint or after executing a single step.
- The display of the value of a tag is supported for tags of the following data types:
  - BOOL
  - DINT
  - UDINT
  - DWORD
  - LREAL
  - TO_Struct_Ipr_Position
  - TO_Struct_Ipr_Frame
  - TO_Struct_Ipr_AxPosition
  - TO_Struct_Ipr_JtPosition

Requirements for changing a tag

- It must be possible to display the value of the tags.
- Changing the value of a tag is supported for the following tags:
  - Local Interpreter variables (VAR, VAR_TEMP, VAR_INPUT, VAR_OUTPUT, return values of functions)
  - Global Interpreter variables (VAR_IPR)
  - Mapped tags which are not defined as "Read only" (VAR_PLC)
  - Tags of the mapped technology objects (except RON)
  - Tags of the Data block technology object of the Interpreter technology object (except RON)

Displaying tag values

If you mouse over the tag in the program editor, a debug tip appears in addition to the regular tooltip. The debug tip shows the name and online value of the tag.

NOTE

Online values of the tag

Tags of the Interpreter program indicate the values at the time of program preparation.

When tags of a technology object or a global data block are mapped, the actual values are displayed.

Changing tag values

For an editable tag, the actual value in the debug tip is displayed as the input field.

To change the value of a tag, follow these steps:

1. In the debug tip, click on the online value of the tag in the input field.
2. Enter a new value.
3. Press "Enter".

The program preparation must be restarted for program execution to be continued with the changed tag value.

NOTE

Automatic restart of program preparation at a change to tag values

As soon as you have changed the value of a tag, the program execution always continues with a restart of the program preparation.

"Continue program execution without a program preparation restart" is not possible. The corresponding buttons in the toolbar display a warning symbol and automatically result in a restart of the program preparation.

Example

You have programmed an IF instruction in your user program.

MCL

```
...
VAR
condition:Bool;
PosA:TO_Struct_Ipr_Position;
PosB:TO_Struct_Ipr_Position;
END_VAR
...
IF condition
THEN linAbs (PosA);
ELSE linAbs (PosB);
```

```
END_IF;
...
```

You have entered a breakpoint before the IF instruction. In "Debug" program mode, the program execution is interrupted at the breakpoint. If you navigate to the "condition" tag with the mouse, the debug tip displays the value of the tag with the existing program preparation.

If the value is "TRUE", the positioning job with linear path motion would be executed to "PosA".

Click the input field in the debug tip and enter "FALSE" to test the positioning job with linear path motion to "PosB". Confirm with the Enter key. Resume program execution.

6.6.8 Stop "Debug" program mode (S7-1500T)

⚠ WARNING

Stop "Debug" program mode

You set the conditions for termination as soon as you activate the "Debug" program mode. Program execution may resume unexpectedly when switching to the "Automatic" program mode.

Possible effects:

- Human life or health may be endangered.
- The machine or the entire plant may be damaged.

Carry out the following protective measures:

- Make sure that you have set the appropriate behavior when activating the "Debug" program mode.

The "Debug" program mode is ended in the following cases:

- Click the "End "Debug" program mode" button in the toolbar of the programming editor. The behavior set in override mode occurs.
- The online connection to the CPU is interrupted for longer than the monitoring time.
- Click the "Monitoring on/off" button in the toolbar of the program editor.
- Click the "Monitor program execution" button in the toolbar of the program editor.

Override mode

When activating the "Debug" program mode, you can define how the Interpreter behaves when exiting the "Debug" program mode:

- Continue program execution

  The Interpreter switches to the "Automatic" program mode and automatically continues the program execution on the program line, at which the "Debug" program mode is finished. Program execution is also continued if program execution was interrupted in Debug mode.

- Abort program execution
  - If the execution of the Interpreter program is interrupted at the time of finishing, the interpreter switches to "Automatic" program mode and aborts the Interpreter program at the current program line. The program status remains at "Interrupted".
  - When the Interpreter program is executed, the Interpreter switches to the "Automatic" program mode and automatically continues the program execution on the program line at which the "Debug" program mode is finished.

Timeout mode

When activating the "Debug" program mode, you can define how the Interpreter behaves when the online connection is lost:

- Stop "Debug" program mode

  The Interpreter switches to "Automatic" program mode. Active "MC_RunProgram" jobs are aborted with a technology alarm. Active motions are stopped with maximum dynamics.

- Continue "Debug" program mode:
  - If the execution of the interpreter program is interrupted at the time of the timeout, the interpreter switches to "Automatic" program mode. The program status remains at "Interrupted".
  - When the Interpreter program is executed, the active job or the active motion sequence is ended and the "Automatic" program mode is activated. Active "MC_RunProgram" jobs are aborted with a technology alarm. The program status is set to "Interrupted".

  You can re-activate the "Debug" program mode and continue running the Interpreter program on the current program line.

Creating MCL programs (S7-1500T) 7

Motion Control Language (MCL) is an interpretive programming language for specifying motion jobs.

The language is based on the Structured Text (ST) programming language specified in DIN EN-61131-3 (IEC 61131-3).

Main characteristics

MCL executes instructions through an Interpreter.

An Interpreter is a program that executes instructions of a programming or script language directly without first compiling it into a machine language program.

The structure of MCL contains a number of motion and technology-specific extensions.

7.1 Syntax of MCL (S7-1500T)

7.1.1 Character set (S7-1500T)

In MCL, the set of permitted characters consists of a subset of the standard ASCII character set:

- Lower and upper-case letters from A to Z
- Arabic numerals 0 to 9
- Control characters (ASCII value 1-31)
- Space (ASCII value 32)
- Special characters

No distinction is made between upper-case and lower-case letters.

MCL interprets comments and symbolic identifiers of PLC data as Unicode characters in UTF-8 format.

Special characters

The following table shows the special characters used in MCL with their descriptions:

Character	Lexical rules	Syntax rules
:	• Delimiters between units of time • Part of the assignment operator	• Delimiter to indicate the type • After the designation before the instructions: – Declaring variables – Functions – CASE instruction

Character	Lexical rules	Syntax rules
.	• Delimiters for representing floating-point numbers • Time interval display • Absolute addressing	Access to a structured variable within a structure.
" "	Introduction for symbol (only for PLC data)	
_	Delimiters in numerical values (can appear in identifiers)	
$	• Access to variables from the Interpreter technology object, and the assigned Kinematics technology object and their kinematics axes • System variable escape characters for specifying control or surrogate characters in strings	
;		Completion of an MCL instruction: • Variable declaration • Instruction part • Constant block • Jump label block • Component declaration
,		Delimiters for lists and jump label blocks variable declaration: • Array data type • Array initialization • Function parameters • List of values
..		Area description: • Array data type • List of values
( )	Use in the comment block	• In expressions for formulating the execution sequence • Function calls • Assignment of structure values
[ ]		• Array declaration • Aggregation of arrays
(**)		Comment block
//		Line comment
%		Introduction of direct identifiers
+		Addition operator positive sign (unary operation)
-		Subtraction operator negative sign (unary operation)

7.1.2 Identifier (S7-1500T)

The lexical rules for constructing identifiers comply with IEC standard 61131-3.

Identifiers are a combination of letters, numbers, and underscores, and must begin with an underscore or letter. The only exception is special literals of the AXIS_OBJECT system type that begin with the character '$'.

With predefined or free identifiers, MCL does not distinguish between upper-case and lower-case letters. Using invalid identifiers results in a runtime error.

Classes

The following table shows the classification of the identifiers into different classes according to their meaning:

Type of identifier	Example
Keywords	Control instructions DO, WHILE
Predefined names for default data types	BOOL, DINT, UDINT
Names for predefined standard functions	ABS, MOD
Predefined names for default constants	TRUE, FALSE, $A1
Freely selectable name	Name of: • Declared variables • Structure components • Parameters • Declared constants • Jump labels

Free identifiers

The structure of free identifiers is defined by the above rules.

Example

The following example shows some free identifiers:

MCL

```
myVar, _myVar, _my_Var
```

Reserved default identifiers

Unused keywords are blocked to ensure the compatibility of Interpreter programs after possible extensions of the MCL syntax. Using reserved identifiers in MCL programs results in a syntax error. The following types of identifiers are syntactically reserved:

- Block names/block keywords
- Timer identifier
- Counter identifier
- Identifiers for operands

Example

The following examples show reserved default identifiers:

MCL

```
Timer
Left
Lower_Bound
```

7.1.3 Literals (S7-1500T)

Literals are constants of elementary data types whose values appear in the text of a program. The following literals are available:

- Numerical literals
- Time literals
- System literals

For numerical data types, the corresponding literals can optionally be explicitly typed, i.e. specification is possible with a preceding data type.

Numerical literals

The following table shows the numerical literals with examples and comments:

Type of literal	Example	Comment
Boolean literal	TRUE, FALSE	-
	BOOL#TRUE, BOOL#FALSE	With type designation
Integer literal	255, +12, -45, 123_456	Decimal integer
	DINT#456_321, UDINT#123_456	With type designation
Floating-point literal	-41_321.0, -0.432, 12.34	-
	1.234e-2, -1.234e-2 5.0E10, 5.0e10, 5E10 3.45E+6, 3.45e+6	With exponents
Binary literal	2#1111_1111, 2#11111111	Equivalent to 255 in the decimal system
Octal literal	8#377	Equivalent to 255 in the decimal system

Type of literal	Example	Comment
Octal literal	8#255	Equivalent to 173 in the decimal system
	DINT#8#255	With type designation
Hexadecimal literal	16#FF, 16#ff	Equivalent to 255 in the decimal system
	16#E0, 16#e0	Equivalent to 224 in the decimal system
	INT#16#FF	With type designation

Time literals

The following table shows the time literals with examples and comments:

Type of literal	Example	Comment
Time duration	T#25ms T#25.8s T#5h10s TIME#5h10s	Type designation required. Can only be used with "waitTime()" and synchronous actions

System literals

The following table shows the system literals with examples and comments:

Type of literal	Example	Comment
AXIS_OBJECT literal	NULL, $A1, $A2, $A3, $A4, $A5, $A6	NULL - Object missing $A1 - $A6 can be used to access the associated kinematics axes of the connected kinematics.
Interpreter literal	$IPR	Access to Interpreter technology object data block variables
Kinematics literal	$KIN	Access to technology object data block variables of the connected kinematics

7.1.4 Expressions (S7-1500T)

Expressions consist of operands and operators.

Operands are constants, tags, and instruction calls.

The operators differentiate between the following expressions:

- Arithmetic expressions
- Logical expressions
- Comparative expressions

The expressions are used in the calculation of operands or independently. The result of the expression is otherwise discarded, for example, for instruction calls without a return value.

Examples

The following example shows some expressions:

MCL

```
// Example for expressions
myVar // operand (simple tag)
sqrt( 27 ) // standard function
FC10( myVar ) // function call
myVar1 AND myVar2 // logical expression
myLrealVar <= 1.0e-2 // comparative expression
-5 + myConst * (3 - 8/5) // simple expression
```

7.1.5 Comments (S7-1500T)

Comments are used to document an MCL program. They have no meaning for the execution of the program and are ignored by the interpreter.

MCL distinguishes between line and block comments. Line comments start with "//" and finish at the end of the line. Block comments begin with "(*" and end with "*)". You can use line comments within block comments. Block comments within row comments are ignored.

Example

The following example shows some comments in MCL:

MCL

```
(*
This program is written in MCL.
*)
myVar1 := 0; // this is a line comment
```

7.1.6 Additional MCL properties (S7-1500T)

The following additional conditions apply:

- It is only possible to create your own variables in the declaration subsections of the respective program organization unit provided for this purpose.
- Motion jobs or technology functions have no return value. Parameters consist of required and optional elements. For optional elements, either a default value or a modal behavior is defined.
- Support of modal parameters, e.g. for specifying dynamics. Modal parameters are parameters that are initialized at the beginning of an MCL program and which retain their value until they are changed by a "set" instruction in the MCL program. Therefore, modal parameters do not need to be specified on the MCL instruction. The Interpreter then uses the value from the modal parameter.
- Synchronous actions that run parallel to motions, i.e. set outputs, perform calculations, etc., are marked separately as such actions.

  An overview of synchronous actions and more information can be found in the section "Synchronous actions (Page 141)".
- The detection and handling of runtime errors by the Interpreter is not intended, i.e. the EN/ENO mechanism used on the PLC side is not supported. A runtime error in the Interpreter leads to an alarm message and cancellation of the Interpreter program.

See also

Synchronous actions (Page 141)

7.2 Data types (S7-1500T)

7.2.1 Overview of data types in MCL (S7-1500T)

Description

A data type is the combination of value ranges and operations in a single unit.

The data type determines how the value of a variable or constant is used in a source program:

- Type and interpretation of data elements
- Permitted ranges for data elements
- Permitted, executable operations
- Notations of constants

Elementary data types

Elementary data types define the structure of data elements that cannot be divided into smaller units. They correspond to the definition in the DIN EN 1131-3 standard. An elementary data type describes a fixed-length memory area and represents bit data types and numerical data types in MCL.

Composite data types

Composite data types describe data types that consist of a fixed number of components.

For the array data type, all components are of the same data type.

For the STRUCT data type, the data types of the components can differ.

The TO_STRUCT_Ipr_* data types describe position specifications of a kinematics or of a coordinate system.

7.2.2 Bit data types (S7-1500T)

Bit data types are combinations of bits that occupy either 1 bit (BOOL data type) or 32 bits. You cannot specify a numerical value range for Double Word. These are bit combinations that can only be used to formulate Boolean expressions.

Type	Keyword	Bit	Value range
Bit	BOOL	1	FALSE, TRUE
Double Word	DWORD	32	16#0000_0000 ... 16#FFFF_FFFF

7.2.3 Numerical data types (S7-1500T)

Numerical data types are available for processing numerical values (for example, for calculating arithmetic expressions).

Type	Keyword	Bit	Value range
Double integer	DINT	32	-2147483648 ... 2147483647
Unsigned double integer	UDINT	32	0 ... 4294967295
Floating point number (double precision)	LREAL	64	-1.79e+308 ... 1.79e+308

7.2.4 Array data type (S7-1500T)

Description

Arrays have a certain number of components of a data type. Arrays with a fixed number of elements of a uniform data type are possible in MCL. All data types (including STRUCT but not ARRAY) are permitted for array components. Only one-dimensional Arrays may be declared.

Syntax

MCL

```
<Var_Name> : ARRAY <[index1..indexN]> OF <Type>;
```

Syntax element	Description
`Var_Name`	Symbolic name of array variable. Free identifier. The name must be followed by the symbol ":".
`ARRAY`	This keyword starts this data type declaration.
`[index1..indexN]`	The smallest (index1) and highest (indexN) possible index (index range). The index can have any integer value (-2147483648 to 2147483647). The index range must be delimited by square brackets.
`OF`	Keyword
`Type`	Specifies the data type of the array components. The following may be used: • Bit data types • Numerical data types • STRUCT • Predefined technology types on the system side The type of the array variables must be followed by the symbol ";"

Example

The initialization of Array variables is carried out in the following example:

MCL

```
VAR CONSTANT
     LIMIT : UDINT := 4;
END_VAR
VAR
     myArr1:ARRAY[1..5] OF LREAL;
     myArr2:ARRAY[1..LIMIT] OF DINT;
     myVar:LREAL;
END_VAR
myVar := myArr1[2];
myVar := myArr2[4];
```

7.2.5 STRUCT data type (S7-1500T)

The STRUCT data type describes a range that comprises a fixed number of components, which can be of different data types. Specifying these data elements occurs immediately after the keyword STRUCT in the component declaration.

The main characteristic of the STRUCT data type is that a data element can also be complex. This means that nesting of STRUCT data types is permitted. The maximum nesting depth is 8.

Syntax and declaration

A structure involves defining the individual components between the keywords STRUCT and END_STRUCT.

There are two ways to use the STRUCT data type:

- Anonymous structure definition in the variable declaration unit
- Structure definition as a user-defined global data type of the Interpreter

Anonymous structure definition in the variable declaration unit

The variables of a newly declared user data type are defined in the variable declaration part of the main program or in functions. All data types are permitted for the components of the structure. Optionally, the components can be initialized with a default value when the structure is defined.

The anonymous structure must be defined in the declaration unit for the declaration of local static variables - VAR - END_VAR.

MCL

```
VAR
  <VAR_Name> : STRUCT
      <Comp1_name> : <Type> [:= <InitValue_1>];
      <Comp2_name> : <Type> [:= <InitValue_2>];
      .....
      <CompN_name> : <Type> [:= <InitValue_N>];
   END_STRUCT;
END_VAR
```

Syntax element	Description	
VAR_Name	The symbolic name of the structured variables is a free identifier. The name is not case-sensitive. The name must be followed by the ":" symbol.	
STRUCT	Definition of the STRUCT data type with the keyword STRUCT.	
<Comp1_name> : <Type> [:= <InitValue_1>]; <Comp2_name> : <Type> [:= <InitValue_2>]; .... <CompN_name> : <Type> [:= <InitValue_N>];	Declaration part of the structure. The structure components (variables or other structures) are declared in this part. Variables can be specified with start values.	
	Comp1_name ... CompN_name	The symbolic name of the structure component is a free identifier. The name is not case-sensitive. The name must be followed by the ":" symbol.

Syntax element	Description	

`<Comp1_name> : <Type> [:= <InitValue_1>];`
`<Comp2_name> : <Type> [:= <InitValue_2>];`
`.....`
`.....`
`<CompN_name> : <Type> [:= <InitValue_N>];`
	`InitValue_1`
	`...`
	`InitValue_N`
		Initialized value of the structure component.
		The value must be followed by the ";" symbol.

`Type`	Data type of the structure component. The following may be used: • Bit data types • Numerical data types • Array data types • STRUCT data types The ";" symbol must appear at the end of the data type.	
`END_STRUCT`	The declaration of the STRUCT data type ends with this keyword. This keyword must be followed by the ";" symbol.	

Example of an anonymous structure definition in the variable declaration unit

MCL

```
//Anonymous structure definition in the tag declaration:
PROGRAM Main
 VAR
     myMotor : STRUCT
          Data : STRUCT
               Current : LREAL;
               Tension : DINT := 5;//declaration with initial value
          END_STRUCT;
     END_STRUCT;
 END_VAR
// usage in statement part - set values
myMotor.Data.Current := 12.35;
myMotor.Data.Tension := 3;
...
END_PROGRAM
```

Structure definition as a user-defined global data type of the Interpreter

A new global data type can be used after its declaration in the data type declaration unit - TYPE - END_TYPE. The programming of this declaration unit must take place outside the main program (PROGRAM - END_PROGRAM block) and is only permitted once per MCL program.

MCL

```
TYPE
   <Type_Name> : STRUCT
      <Comp1_name> : <Type> [:= <InitValue_1>];
      <Comp2_name> : <Type> [:= <InitValue_2>];
      .....
      <CompN_name> : <Type> [:= <InitValue_N>];
   END_STRUCT;
END_TYPE
```

Syntax element	Description	
`Type_Name`	Symbolic name of the global structured data type. Free identifier. The name is not case-sensitive. The name must be followed by the ":" symbol.	
`STRUCT`	Definition of the STRUCT data type with the keyword STRUCT.	

`<Comp1_name> : <Type> [:= <InitValue_1>];`
`<Comp2_name> : <Type> [:= <InitValue_2>];`
`.....`
`.....`
`<CompN_name> : <Type> [:= <InitValue_1>];`
	Declaration part of the structure.
	The structure components (variables or other structures) are declared in this part.	

	`Comp1_name`
	`...`
	`CompN_name`
		The symbolic name of the structure component is a free identifier.
		The name is not case-sensitive.
		The name must be followed by the ":" symbol.

	`InitValue_1`
	`...`
	`InitValue_N`
		Initialized value of the structure component.
		The value must be followed by the ";" symbol.

`Type`	Data type of the structure component. The following may be used: • Bit data types • Numerical data types • Array data types • STRUCT data types The ";" symbol must appear at the end of the data type.	
`END_STRUCT`	The declaration of the STRUCT data type ends with this keyword. This keyword must be followed by the ";" symbol.	

Before using a variable with a user-defined data type, you must declare a new variable with the same data type in a variable declaration unit. You can find more information in the section "Declaring variables and function parameters (Page 96)".

To access a specific element of a structure (read or write), it is necessary to use the "." symbol:

```
<VAR_Name>.<Comp2_name> := <Value>;
```

Example of structure definition as a user-defined global data type of the Interpreter

MCL

```
//Structure definition as user defined interpreter global data type:
TYPE
     Motor : STRUCT
          mType : DINT := 5; // declaration with initial value
          current : LREAL;
     END_STRUCT;

     Gear : STRUCT
          gType : DINT;
          ratio : LREAL;
     END_STRUCT;
END_TYPE
PROGRAM Main
 VAR
     myMotor : Motor;
     myGear : Gear;
 END_VAR
```

MCL

```
// usage in statement part -set values
myMotor.current := 2.35;
myGear.gType := 7;
...
END_PROGRAM
```

7.2.6 Data type TO_Struct_Ipr_Position (S7-1500T)

Description

The TO_Struct_Ipr_Position data type is used to define a position in the world coordinate system or object coordinate system.

You can find more information about kinematics types, motion types, coordinate systems, and frames, as well as kinematics axes in the "S7-1500T Kinematics functions (Page 10)" documentation.

Variables

The data type consists of the following variables:

Variable	Data type	Description
x	LREAL	Position offset in x direction of the reference coordinate system
y	LREAL	Position offset in y direction of the reference coordinate system
z	LREAL	Position offset in z direction of the reference coordinate system
A	LREAL	Rotation (Euler angle of the first rotation) of the position about the z-axis of the coordinate system. The single-rotated coordinate system consists of the axes x', y', and z
B	LREAL	Rotation (Euler angle of the second rotation) of the position about the y' axis of the coordinate system. The double-rotated coordinate system consists of the axes x'', y', and z'
C	LREAL	Rotation (Euler angle of the third rotation) of the position about the x'' axis of the coordinate system. The triple-rotated coordinate system consists of the axes x'', y'' and z''. The x'', y'', and z'' axes correspond to the x, y, and z axes of the rotated coordinate system

Use

The value range of each variable of this data type depends on the type of kinematics, the type of frame, and the instruction in which this variable is used.

Instruction	Parameter name	Description
linAbs	<pos> : TO_Struct_Ipr_Position;	Linear motion with absolute position specification
circRel	<pos> : TO_Struct_Ipr_Position; [auxPos:] TO_Struct_Ipr_Position;	Circular motion with relative target position
ptpAbs	<pos> : TO_Struct_Ipr_Position;	sPTP (synchronous point-to-point) motion with absolute target position in Cartesian coordinates

Example

MCL

```
PROGRAM Main
 VAR
    myPos1 : TO_Struct_Ipr_Position;
    myPos2 : TO_Struct_Ipr_Position :=
   ( x := 10.01, y := 20.02, z := 30.03,
     A := 0.0, B := 0.0, C := 0.0 );
 END_VAR
// set parameters for absolute position
myPos1.x := 10.5;
myPos1.y := -123.84;
myPos1.z := 17.35;
myPos1.A := 0.0;
myPos1.B := 0.0;
myPos1.C := 0.0;
(* usage of myPos1 and myPos2 variables of TO_Struct_Ipr_Position data type
as function parameter *)
linAbs( myPos1 );
linAbs( myPos2 );
...
END_PROGRAM
```

Default values

The pre-initialization of the variables depends on the context used. By default, the following rules apply to uninitialized components for variables and units:

Type	Use context	Default values
TO_Struct_Ipr_Position	Variable	x = 0.0 y = 0.0 z = 0.0 A = 0.0 B = 0.0 C = 0.0
	Unit as relative position specification in a technological instruction	
	Unit as absolute position specification in a technological instruction	Actual value that the variable has in the program lead of the Interpreter program execution

7.2.7 Data type TO_Struct_Ipr_AxPosition (S7-1500T)

Description

The TO_Struct_Ipr_AxPosition data type is a variable type for defining the position of up to 6 axes in the machine coordinate system (MCS).

You can find more information about kinematics types, motion types, coordinate systems, and frames, as well as kinematics axes in the "S7-1500T Kinematics functions (Page 10)" documentation.

Syntax

The data type consists of the following variables:

Identifier	Data type	Description
a1	LREAL	Position of the kinematics axis A1 in the MCS (axis coordinates)
a2	LREAL	Position of the kinematics axis A2 in the MCS (axis coordinates)
a3	LREAL	Position of the kinematics axis A3 in the MCS (axis coordinates)
a4	LREAL	Position of the kinematics axis A4 in the MCS (axis coordinates)
a5	LREAL	Position of the kinematics axis A5 in the MCS (axis coordinates)
a6	LREAL	Position of the kinematics axis A6 in the MCS (axis coordinates)

Use

The MCS contains the position data of the connected kinematics axes and thus combines the axis positions as an array or vector.

The value range of each variable of this data type depends on the type of kinematics, the type of frame, and the instruction in which this variable is used.

The TO_Struct_Ipr_AxPosition data type is used for data types of variables which can be parameters of the following technological instructions:

Instruction	Parameter name	Description
ptpAxAbs	<AxPos> : TO_Struct_Ipr_AxPosition;	sPTP (synchronous point-to-point) Motion with absolute target position in machine coordinates
ptpAxRel	<AxPos> : TO_Struct_Ipr_AxPosition;	sPTP (synchronous point-to-point) Motion with relative target position in machine coordinates

Example

MCL

```
PROGRAM Main
 VAR
   myPos1, myPos2 : TO_Struct_Ipr_AxPosition;
 END_VAR
// set parameters for target position in machine coordinate system (MCS)
 myPos1.a1 := 10.5;
 myPos1.a2 := -123.84;
 myPos1.a3 := 17.35;
 myPos1.a4 := 2.0;
 myPos1.a5 := -13.5;
 myPos1.a6 := 10.34;
 ptpAxAbs( myPos1 );
// set parameters for relative target position in machine coordinates (MCS)
 myPos2.a1 := 30.2;
 myPos2.a2 := -13.64;
 myPos2.a3 := 1.78;
 myPos2.a4 := -5.1;
 myPos2.a5 := -12.5;
 myPos2.a6 := 50.34;
// usage of myPos2 tag of TO_Struct_Ipr_AxPosition data type as function parameter
 ptpAxRel( myPos2 );
END_PROGRAM
```

Default values

The pre-initialization of predefined technological system structure data types depends on the context used. By default, the following rules apply to uninitialized components for variables and units:

Type	Use context	Default values
TO_Struct_Ipr_AxPosition	Variable	a1 = 0.0 a2 = 0.0 a3 = 0.0 a4 = 0.0 a5 = 0.0 a6 = 0.0
	Unit as relative position specification in a technological instruction	
	Unit as absolute position specification in a technological instruction	Actual value that the variable has in the program lead of the Interpreter program execution

7.2.8 Data type TO_Struct_Ipr_JtPosition (S7-1500T)

Description

The TO_Struct_Ipr_JtPosition data type is a variable type for defining the position of kinematics axes in the joint coordinate system (JCS) with up to 6 degrees of freedom.

You can find more information about kinematics types, motion types, coordinate systems, and frames, as well as kinematics axes in the "S7-1500T Kinematics functions (Page 10)" documentation.

Syntax

The data type consists of the following variables:

Identifier	Data type	Description
j1	LREAL	Position of the 1st kinematics axis in the JCS
j2	LREAL	Position of the 2nd kinematics axis in the JCS
j3	LREAL	Position of the 3rd kinematics axis in the JCS
j4	LREAL	Position of the 4th kinematics axis in the JCS
j5	LREAL	Position of the 5th kinematics axis in the JCS
j6	LREAL	Position of the 6th kinematics axis in the JCS

Use

A kinematics obtains Cartesian coordinates from different joint positions of the kinematics axes. The joint coordinate system (JCS) represents the mechanical positions of the kinematics joints. Joints can perform linear or rotary motions.

The value range of each variable of this data type depends on the type of kinematics, the type of frame, and the instruction in which this variable is used.

The TO_Struct_Ipr_JtPosition data type is used for data types of variables which can be parameters of the following technological instructions:

Instruction	Parameter name	Description
ptpJtAbs	<jtPos> : TO_Struct_Ipr_JtPosition;	sPTP motion with absolute target specification in joint coordinates
ptpJtRel	<jtPos> : TO_Struct_Ipr_JtPosition;	sPTP motion with relative target specification in joint coordinates

Example

MCL

```
PROGRAM Main
 VAR
   myPos: TO_Struct_Ipr_JtPosition;
 END_VAR
// set parameters for position in joint coordinates system (JCS)
 myPos.j1 := 10.5;
 myPos.j2 := -123.84;
 myPos.j3 := 17.35;
 myPos.j4 := 2.0;
 myPos.j5 := -13.5;
 myPos.j6 := 10.34;
// usage of myPos tag of TO_Struct_Ipr_JtPosition data type as function parameter
 ptpJtAbs( myPos );
 ptpJtRel( myPos );
END_PROGRAM
```

Default values

The pre-initialization of predefined technological system structure data types depends on the context used. By default, the following rules apply to uninitialized components for variables and units:

Type	Use context	Default values
TO_Struct_Ipr_JtPosition	Variable	j1 = 0.0 j2 = 0.0 j3 = 0.0 j4 = 0.0 j5 = 0.0 j6 = 0.0
	Unit as relative position specification in a technological instruction	
	Unit as absolute position specification in a technological instruction	Actual value that the variable has in the program lead of the Interpreter program execution

7.2.9 Data type TO_Struct_Ipr_Frame (S7-1500T)

Description

Frames indicate the offset and rotation of one coordinate system relative to another coordinate system. The data type TO_Struct_Ipr_Frame can be a variable type for this specification. You can find more information about kinematics types, motion types, coordinate systems, and frames, as well as kinematics axes in the "S7-1500T Kinematics functions (Page 10)" documentation.

Syntax

The data type consists of the following variables:

Identifier	Data type	Description
x	LREAL	Moves the frame in the x direction of the reference coordinate system
y	LREAL	Moves the frame in the y direction of the reference coordinate system
z	LREAL	Moves the frame in the z direction of the reference coordinate system
A	LREAL	Rotation (Euler angle of the first rotation) of the frame about the z-axis of the coordinate system. The single-rotated coordinate system consists of the axes x', y', and z
B	LREAL	Rotation (Euler angle of the second rotation) of the frame about the y' axis of the coordinate system. The double-rotated coordinate system consists of the axes x'', y', and z'
C	LREAL	Rotation (Euler angle of the third rotation) of the frame about the x'' axis of the coordinate system. The triple-rotated coordinate system consists of the axes x'', y'', and z''. The x'', y'', and z'' axes correspond to the x, y, and z axes of the rotated coordinate system or of the rotated zone

Use

The value range of each variable of this data type depends on the type of kinematics, the type of frame, and the instruction in which this variable is used.

The TO_Struct_Ipr_Frames data type is used for data types of variables which can be parameters of the following technological instructions:

Instruction	Parameter name	Description
defOcs	<frame> : TO_Struct_Ipr_Frame;	Defines one of the three object coordinate systems (OCS)
defTool	<frame> : TO_Struct_Ipr_Frame;	Defines one of the tool coordinate systems (TCS)

Instruction	Parameter name	Description
trackIn	<origin> : TO_Struct_Ipr_Frame; <initPos> : TO_Struct_Ipr_Frame;	Provides the functionality for conveyor synchronization
defWsZone	<fr> : TO_Struct_Ipr_Frame;	Definition of work areas
defKinZone	<fr> : TO_Struct_Ipr_Frame;	Definition of the kinematics range

Example

MCL

```
PROGRAM Main
 VAR
 //coordinates for OCS frame
 myPos_frOCS: TO_Struct_Ipr_Frame := ( x := 10.01, y := 20.02, z := 30.03,
                                       A := 50.0, B := 30.0, C := 45.0 );
 //coordinates for TCS frame
 myPos_frTool: TO_Struct_Ipr_Frame := ( x := 20.01, y := 50.02, z := 60.03,
                                        A := 7.0, B := 10.0, C := -45.0 );
 END_VAR
// function defines the position of OCS in relation to WCS, using myPos_frOCS tag
defOcs( 1, myPos_frOCS );
// function defines TCS, using myPos_frTool tag
defTool( 1, myPos_frTool );
...
END_PROGRAM
```

Default values

By default, the following rules apply to uninitialized components for variables and units:

Type	Use context	Default values
TO_Struct_Ipr_Frame	-	x, y, z, A, B, C = 0.0

7.2.10 Data type AXIS_OBJECT (S7-1500T)

The data type AXIS_OBJECT establishes the internal reference to a kinematics axis or to a single axis.

Keyword	Value range
AXIS_OBJECT	References to the positioning axes and synchronized axes connected to the kinematics ("kinematics axis"): $A1, $A2, $A3, $A4, $A5, $A6. Mapped single axes are accessed using the identifiers defined in the mapping rule. The default value for uninitialized variables of type AXIS_OBJECT is "NULL", which means that the variable points to no axis.

You can find more information on the AXIS_OBJECT data type in the documentation "S7-1500T Kinematics Functions (Page 10)".

7.2.11 Data type conversions (S7-1500T)

In MCL, the data type compatibility of the operands must be taken into account when processing instructions and expressions and when transferring parameters to functions.

When linking two values of different data types or assigning expressions to variables, the mutual compatibility of the data types involved must always be checked.

The following cases lead to incorrect results:

- Switching to another type class, such as from a bit data type to a numerical data type.
- A switch within the same type class, if not loss-free, i.e. while maintaining the value and the precision.

MCL distinguishes between the following data type conversions:

- Implicit data type conversion
- Explicit data type conversion

Implicit data type conversion

If the operands are compatible, automatic conversion takes place according to defined conversion rules.

The MCL conversion rules apply to implicit conversion of data types. Implicit conversion is possible for the source data types DINT and UDINT.

The value must be transferable to the target data type. There is no sign conversion.

The following table shows possible (✓) implicit type conversions:

		Source				
		BOOL	**DWORD**	**DINT**	**UDINT**	**LREAL**
Target	**BOOL**		—	—	—	—
	DWORD	—		—	—	—
	DINT	—	—		—	—
	UDINT	—	—	—		—
	LREAL	—	—	✓	✓	

Example

The following example shows an implicit type conversion:

MCL

```
PROGRAM Main
 VAR
     Value_1 : DINT;
     Value_2 : UDINT;
     Value_3 : LREAL;
 END_VAR
// Value_1 is implicitly converted to a tag of data type LREAL
     Value_3 := Value_1;
// Value_2 is implicitly converted to a tag of data type LREAL
     Value_3 := Value_2;
 ...
END_PROGRAM
```

Explicit data type conversion

Operands with incompatible data types can be converted explicitly using corresponding standard functions.

The following table shows functions for the explicit data type conversion for BOOL, DWORD, DINT, UDINT and LREAL:

		Source				
		BOOL	**DWORD**	**DINT**	**UDINT**	**LREAL**
Target	**BOOL**		DWORD_TO_BOOL	DINT_TO_BOOL	UDINT_TO_BOOL	—
	DWORD	BOOL_TO_DWORD		DINT_TO_DWORD	UDINT_TO_DWORD	—
	DINT	BOOL_TO_DINT	DWORD_TO_DINT		UDINT_TO_DINT	LREAL_TO_DINT
	UDINT	BOOL_TO_UDINT	DWORD_TO_UDINT	DINT_TO_UDINT		LREAL_TO_UDINT
	LREAL	—	—	DINT_TO_LREAL	UDINT_TO_LREAL	

The following table shows conversion rules for each function:

Source	Target	Explanation	Mnemonic of the instruction
BOOL	DWORD	Only the least significant bit is set in the target data type. The leading bits are set to zero.	BOOL_TO_DWORD
	DINT		BOOL_TO_DINT
	UDINT		BOOL_TO_UDINT
DWORD	BOOL	Copies the least significant bit.	DWORD_TO_BOOL
	DINT	The bit pattern of the source value is transferred unchanged and right-justified to the target data type.	DWORD_TO_DINT
	UDINT		DWORD_TO_UDINT
DINT	BOOL	Copies the least significant bit.	DINT_TO_BOOL
	DWORD	The bit pattern of the source value is transferred unchanged and right-justified to the target data type.	DINT_TO_DWORD
	UDINT	The bit pattern of the source value is converted and transferred to the target data type.	DINT_TO_UDINT
	LREAL	The value is converted to the format of the target data type.	DINT_TO_LREAL[1)]
UDINT	BOOL	Copies the least significant bit.	UDINT_TO_BOOL
	DWORD	The bit pattern of the source value is transferred unchanged and right-justified to the target data type.	UDINT_TO_DWORD
	DINT		UDINT_TO_DINT
	LREAL	The value is converted to the format of the target data type. (The value "1" is converted to the value "1.0", for example.)	UDINT_TO_LREAL[1)]
LREAL	DINT	The value is converted to the target data type. Depending on the value of the floating-point number, the following conversion rules apply: • < -2147483648.0: The target value -2147483648 is output. • -2147483648.0 ... 2147483647.0: The integer part of the floating-point number is output as a target value. The decimal places are truncated, e.g. "1.9" becomes "1". • > 2147483647.0: The target value 2147483647 is output.	LREAL_TO_DINT[1)]

Source	Target	Explanation	Mnemonic of the instruction
LREAL	UDINT	The value is converted to the target data type. Depending on the value of the floating-point number, the following conversion rules apply: • < 0.0: The target value 0 is output. • 0.0 ... 4294967295.0: The integer part of the floating-point number is output as a target value. The decimal places are truncated, e.g. "1.9" becomes "1". • > 4294967295.0: The target value 4294967295 is output.	LREAL_TO_UDINT[1)]
[1)] The conversion can also be made with special conversion functions, see section "Conversions (Page 336)".			

Each instruction for conversion of a data type has only one input parameter.

Example

In the following example, an explicit conversion is necessary because the target data type is a lower-order data type than the source data type.

MCL

```
PROGRAM Main
VAR
  var1 : DINT;
  var2 : LREAL;
END_VAR
// DINT is lower order than LREAL
var1 := LREAL_TO_DINT( var2 );
// . . .
END_PROGRAM
```

7.3 Tags (S7-1500T)

7.3.1 Categories of variables and function parameters (S7-1500T)

Variables

The table shows the variable categories:

Variable	Explanation
Local static variables (VAR)	User-defined local variables whose value is retained during execution of the main program. Access to such variables is only possible from the organization unit of the program in which they are declared.
Local temporary variables (VAR_TEMP)	User-defined local variables whose value is retained only during execution of the function or interrupt routine. Access to such variables is only possible from the organization unit of the program in which they are declared.
Global Interpreter variables (VAR_IPR)	User-defined local variables whose value is retained during execution of the Interpreter program. Access to such variables is possible from any organization unit of the program.

Function parameters

The table shows the function parameter categories:

Function parameter	Explanation
Input parameters (VAR_INPUT)	Input parameters apply the current input values when the function is called.
Output parameters (VAR_OUTPUT)	Output parameters transfer the current output values to the calling program organization unit (main program or other function). You can write data to them and read data from them.

Declaration

Each category of variables or parameters has its own declaration subsection, identified by its own keyword pair. Each subsection contains the declarations that are permitted for that declaration subsection. These subsections can be arranged in any order. More information can be found in the sections "Overview of the MCL program structuring (Page 152)" and "Function parameters (Page 160)".

7.3.2 Declaring variables and function parameters (S7-1500T)

Declaring variables

A variable must be declared before it can be used. A variable declaration consists of a variable name (identifier), a data type specification, and an optional initialization. In MCL, variables are declared in special blocks of the program organization unit. Depending on the variable categories, the following declaration blocks are possible:

- Local static variables (VAR)
- Local temporary variables (VAR_TEMP)
- Global Interpreter variables (VAR_IPR)

You declare the variable name within the variable declaration.

Declaring function parameters

Function parameters are formal parameters of a function. When the function is called, the actual parameters replace the formal parameters, thus forming a mechanism for exchanging data between the called and the calling block.

Depending on the variable category, the following declaration blocks are possible:

- Input parameters (VAR_INPUT)
- Output parameters (VAR_OUTPUT)

You define the function signature, i.e. the call interface of the function.

See also

Overview of the MCL program structuring (Page 152)
Function parameters (Page 160)

7.3.3 Local static variables (S7-1500T)

Description

Static variables are local, user-defined variables whose value is retained when the main program is running.

Syntax

This variable type is declared in the main program of the Interpreter program in the declaration block VAR/END_VAR. Use is optional.

Example

The following example shows how to use local static variables:

MCL

```
VAR
  ……
  myVar : DINT;
  MEASARR : ARRAY [1..10] OF LREAL;
  myFlag : BOOL;
  ……
END_VAR
```

Access

A local static variable can only be accessed from the program organization unit in which it is declared, i.e. only from the main program of the Interpreter.

7.3.4 Local temporary tags (S7-1500T)

Description

Temporary tags are local, user-defined tags that belong locally to a function and do not occupy a static memory area.

Syntax

Local temporary tags are declared in the VAR_TEMP/END_VAR declaration block. This declaration block is part of the function. Use is optional.

Example

The following example shows how to use local temporary tags:

MCL

```
VAR_TEMP
  ......
  BUFFER_1 : ARRAY [1..10] OF DINT ;
  AUX1, AUX2 : LREAL ;
  ......
END_VAR
```

Access

Access to a tag is always from the code section of the function in which the tag is declared in the declaration block (internal access). Temporary tags cannot be accessed from outside the function in which they are declared.

NOTE

The VAR block in the main program and VAR_TEMP in functions are functionally synonymous blocks.

7.3.5 Global interpreter tags (S7-1500T)

Description

Global interpreter tags are visible for each program organization unit of an interpreter program.

Syntax

VAR_IPR/END_VAR is used for the declaration of global interpreter tags. This declaration block must be part of the main program. Use is optional.

Example

The following example shows how to use global interpreter tags:

MCL

```
VAR_IPR
  ......
  glbVar1 : DINT;
  ......
END_VAR
```

Access

These tags can be accessed from the main program and from all functions called from it.

Global Interpreter variables cannot be defined in a function.

Access from the user program to global interpreter variables is not possible.

7.3.6 Input parameters (S7-1500T)

Description

Formal input parameters are assigned the current input values (data flow into the function) of a function. Input parameters apply the current input values when the function is called.

Syntax

The input parameters of functions are declared in the declaration block VAR_INPUT/END_VAR. This declaration block must be part of the function. Use is optional.

Example

The following example shows how to use input parameters:

MCL

```
VAR_INPUT
  ......
  inPar : DINT;
  inTriger : BOOL;
  ......
END_VAR
```

Access

These parameters can only be accessed from the code section of the function in whose declaration unit the parameter is declared. More information on the use of parameters and the associated data exchange can be found in the section "Function parameters (Page 160)".

7.3.7 Output parameters (S7-1500T)

Description

Formal output parameters are used to transfer output values (data flow from the block to the outside). Output parameters transfer the current output values to the calling program organization unit (main program or instructions). Data can be written to and read from them.

Syntax

The parameters of output functions are declared in the VAR_OUTPUT/END_VAR declaration block. This declaration block must be part of the function. Use is optional.

Example

The following example shows how to use output parameters:

MCL

```
VAR_OUTPUT
  ……
  outPar : BOOL;
  outPar_Current : LREAL;
  ……
END_VAR
```

Access

These parameters can be accessed from the code section of the function in whose declaration block the parameter is declared. More information on the use of parameters and the associated data exchange can be found in the section "Function parameters (Page 160)".

7.3.8 Initializing variables and function parameters (S7-1500T)

Description

Variables and function parameters (input and output parameters of a function) receive a start value when they are declared in declaration units. Initialization is performed by a value assignment (symbol set ":=") that follows the data type specification.

Initialization of variables and parameters

To initialize variables or parameters, a value must be assigned to a variable:

MCL

```
myVar: LREAL := 12.25;
a1, a2, a3, b1 : DINT := 11; // tag list initialization
```

When initializing, no expression may be specified as an initialization value. Only values or constants are allowed.

MCL

```
myVar: LREAL := 12.25 + 3.14 * 2.0; // not allowed
```

Initialization of arrays

Several options are available for initializing an array:

- Array aggregation
  `my1dArr2 : ARRAY [1..5] OF DINT := [1, 3, 8, 4, 0];`
- Assignment of an initialization list by using a repetition factor and a value with outer square brackets:
  `myARRAY : ARRAY[1..10] OF LREAL:= [10(3.29)];`
  with "10" as the repetition factor and "3.29" as the value
- Mixed variant, i.e. the aggregation can also contain a repetition factor and value:
  `d : ARRAY[1..12] OF LREAL := [0.0, 5(10.0), 1.0];`

If the number of values in the array initialization exceeds the size of the associated array, a semantic error is generated. If you specify fewer values in the array initialization compared to the size of the array, the remaining values are initialized with the default value of the data type used, as is the case in the mixed variant example above:

- d[1] is 0.0.
- This is followed by 5 times the value 10.0 (d[2] to d[6]).
- d[7] is 1.0.
- As the values 8 to 12 (d[8] to d[12]) are not explicitly initialized, they receive the type default value 0.0.

Initialization of structures

MCL also enables the initialization of variables of user-defined types via structure aggregates. It is possible to initialize any structured element:

MCL

```
// structure with initialisation
myStructVar : STRUCT
  a : LREAL := 12.3;
  b : DINT := 12 ;
  c : ARRAY[1..12] OF LREAL := [0.0, 5(10.0), 1.0];
END_STRUCT ;
```

After initialization, the elements have the following values:

- myStructVar.c[1] = 0.0;
- myStructVar.c[2,3,4,5,6] = 10.0;
- myStructVar.c[7] = 1.0;
- myStructVar.c[8,9,10,11,12] = 0.0;

Example

The following example illustrates how to initialize variables:

MCL

```
TYPE
  Motor : STRUCT
     mType : DINT;
     Data : STRUCT
        Loadcurrent : LREAL;
        //initialization for the structure element
        Voltage : LREAL := 5.0;
     END_STRUCT;
  END_STRUCT;
END_TYPE

PROGRAM Main

//initialization for global interpreter variables
 VAR_IPR
  myVar2 : DINT := 0;
  myVar3, myVar4 : DINT := 0;
 END_VAR

//initialization for local variables
 VAR
  myVar1 : LREAL := 0.0;
   //array variables with optional initialization
   my1dArr2 : ARRAY [1..5] OF DINT := [1, 3, 8, 4, 0];
   //each element of array has the same value = 1.234
  my1dArr3 : ARRAY [1..5] OF LREAL := [5(1.234)];
   myMotor1 : Motor := ( mType := 1,
                         Data := (Loadcurrent := 15.3,
                                  Voltage := 13.5) );
 END_VAR

  //statement part of the main program
  my1dArr3[4] := my1dArr2[1] + my1dArr3[5];
  my1dArr3[5] := 14.67;
  myMotor1.mType := 2;
  myMotor1.Data.Loadcurrent := 8.5;
END_PROGRAM
```

7.3.9 Using variables of the technology object data blocks (S7-1500T)

Description

The Interpreter provides direct access to its own technology object data block variables and to technology object data block variables of mapped technology objects in the MCL program.

The following technology objects are supported:

Technology object	Application
"Interpreter instance" Separate technology object of the Interpreter	$IPR
"Kinematics" Connected Kinematics technology object	$KIN
"Kinematics axis" Technology object axis	$A1, $A2, $A3, $A4, $A5, $A6
References to mapped technology objects (e.g. an axis)	Identifier specified in the mapping file based on the mapping rule

Syntax

A variable is accessed in a structured form by a combination of the scope of the assigned object `(<TO_Identifier>)` and the variable name `(<VarName>)`:

Syntax	Explanation	
"<TO_Identifier>.<VarName>"	<TO_Identifier>	As the technology object, the identifier can be: • Separate technology object "Interpreter instance" • Connected "Kinematics" objects • "Kinematics axis" • References to mapped technology objects (axis) Note: To access these technology objects you must use the "$" symbol in front of the name of the technology object
	"."	Because a technology object has a structure type ("STRUCT"), the variables of the technology objects are accessed as an element of the structure data type with the symbol "."
	<VarName>	Symbolic name of the variables of the Structure technology object

Only directly changeable tags (labeled as "Direct (DIR)") can be written in the technology object data block. Reading and writing of corresponding variables is possible in all value assignments and expressions. It is therefore necessary to indicate the preceding range of validity.

Example

Technology object	Description	Example in MCL

$A1	Kinematics axis A1	//set acceleration value for axis A1
		$A1.DynamicDefaults.Acceleration := 10.0;
		//set velocity value for axis A2
		$A1.DynamicDefaults.Velocity := 10.0;
		//read actual position for axis A3
		posAxisA3 := $A3.ActualPosition;
		//read error status of the drive
		driveError := $A5.StatusDrive.Error;
		//set home-position for axis A6
		$A6.Homing.HomePosition := 33.25;

$A2	Kinematics axis A2	
$A3	Kinematics axis A3	
$A4	Kinematics axis A4	
$A5	Kinematics axis A5	
$A6	Kinematics axis A6	
$IPR	Interpreter	//read Interpreter status word IPR_StatusWord := $IPR.StatusWord;

$KIN	Kinematics	//set limit for jerk
		$KIN.DynamicLimits.Path.Jerk := 10000.0
		//set override of velocity
		$KIN.Override.Velocity := 100.0;

Identifier of the technology object	References to mapped technology objects (for example, individual axis).
	Identifier defined in the assignment rule in the example:
	"myAxis"
		//set TO "myAxis" to absolute position
		posAbs(myAxis, 20.0);
		preHalt(); //Pre-run stop
		//read actual position for TO "myAxis"
		myVar := myAxis.ActualPosition;

The description of all variables for each technology object can be found in the documentation "Variables of the technology object data blocks" and "Variables of the Kinematics technology object".

7.4 Constants (S7-1500T)

7.4.1 Categories of constants (S7-1500T)

Description

Constants are data elements that have a value that cannot be changed during program execution. MCL supports using the following sets of constants, depending on the level of access:

- Local constants: Only visible to the program organization unit in which they are declared.
- Global Interpreter constants: Visible to each program organization unit of an Interpreter program.
- System constants: Visible to each program organization unit of an Interpreter program. System constants are constants provided by the system.

Value assignments and expressions use constants in addition to variables and function parameters.

Constants have a symbolic name. Symbolic names of constants must be declared separately in a separate declaration block.

7.4.2 Declaring constants (S7-1500T)

Description

Constants are declared by type in separate declaration subsections of the program organization unit. The required value setting is made by a value assignment (symbol set ":="), which follows the data type specification. When initializing, no expression may be specified as an initialization value. Only values or constants are allowed.

The following declaration subsections are possible:

- VAR CONSTANT/END_VAR for local constants.
- VAR_IPR CONSTANT/END_VAR for global Interpreter constants.

Global Interpreter constants cannot be defined in a function.

The following table shows which types of constants can be declared in the different program organization units:

Name of the declaration unit	Use	Syntax in MCL	POE	
			Main program	Function

Declaration unit for local constants	Optional	`VAR CONSTANT`
		`.....`
		`END_VAR`
			✓	✓

Declaration unit for global Interpreter constants	Optional	`VAR_IPR CONSTANT`
		`.....`
		`END_VAR`
			✓	

See also

Declaring variables and function parameters (Page 96)

7.4.3 Local constants (S7-1500T)

Syntax

The VAR CONSTANT/END_VAR construct is used to declare local constants. This declaration block must be a part of the main program or function. Use is optional.

Example

The following example shows how to declare constants in the main program and in a function:

MCL

```
PROGRAM Main
  VAR CONSTANT
    myConst : DINT := 20;
  END_VAR
  VAR
    myVar1, myVar2 : LREAL;
  END_VAR
  myVar1 := 1.5;
  myVar2 := 2.3 + myConst;
END_PROGRAM
FUNCTION myFct : VOID
  VAR CONSTANT
    myConst : DINT := 50;
  END_VAR
  VAR
    myVar3 : LREAL;
  END_VAR

  myVar3 := myConst + 4.5;
END_FUNCTION
```

Access

Local constants are only visible to the program organization unit in which they are declared (main program or function).

7.4.4 Global constants (S7-1500T)

Syntax

The VAR_IPR CONSTANT/END_VAR construct is used to declare global interpreter constants in the main program. The use of this declaration block is optional.

Example

The following example shows how to declare a global interpreter constant and use local and global constants:

MCL

```
PROGRAM Main
  VAR CONSTANT
  myConst : DINT := 10;
  END_VAR
  VAR_IPR CONSTANT
    myGlobalConst : DINT := 20;
  END_VAR
  VAR
    myVar1, myVar2 : LREAL;
  END_VAR
  myVar1 := 1.5;
  myVar2 := 2.3 + myConst + myGlobalConst;
END_PROGRAM
FUNCTION myFct : VOID
  VAR CONSTANT
    myConst : DINT := 20;
  END_VAR
  VAR
    myVar3 : LREAL;
  END_VAR

  myVar3 := myConst* myGlobalConst +4.5; // myVar3 = 20 * 20 + 4.5
END_FUNCTION
```

Access

The global interpreter constants are visible for each program organization unit of the interpreter. These can be accessed from the main program and from all functions called from it.

7.4.5 System constants (S7-1500T)

System constants are constants provided by the system.

Overview of system constants

System constant	Data type	Meaning
NULL_POS[1]	TO_Struct_Ipr_Position	The NULL_POS system constant can be used in the "circAbs()" and "circRel()" MCL instructions for the mandatory end position "pos". When the instruction is called with "mode" :=1, only the orientation part of the end position "pos" is relevant. If the orientation does not change during a circular motion, use "pos" as end position and the NULL_POS system constant as "empty spare position".

[1] A declared tag/constant with identical name overwrites the system constant

Example

The following example shows you how the "NULL_POS" system constant is used in the "circAbs()" MCL instruction.

MCL

```
PROGRAM Main
  setDyn ( v := 50, a := 100, d := 200, j := 1000 );
 setOriDyn (v := 100, a := 200, d := 300, j := 1000);
 setPlane (0);
 circAbs ( NULL_POS, auxPos := ( x := 200, y := 0, z := 150, A := 90, B := 0, C := 0 ),
mode := 1, arc := 270.0, cDir := 0);

  //End position NULL_POS only relevant for orientation coordinates with mode 1
END_PROGRAM
```

Access

System constants can be accessed from the main program and from all functions called from it.

7.5 Operators (S7-1500T)

7.5.1 Overview of operators (S7-1500T)

Expressions in MCL include operators. An operator is a rule that can create a new operand from an operand.

Several classes of operators are supported by MCL:

- Assignment operators
- Arithmetic operators
- Relational operators
- Logical operators
- Element selection operators

7.5.2 Assignment operators (S7-1500T)

Description

A value-assigning operator assigns the value of an expression to a variable. The previous value is overwritten. The variable exists only after it has been declared. You can find more information in the section "Declaring variables and function parameters (Page 96)".

Syntax

The expression is evaluated on the right side of the assignment symbol of the operator. The result is stored in the variable, whose name is on the left side of the assignment operator (target variable).

The following table shows the syntax rules for assignment operators:

Syntax	Designation	
`<VAR_Name> := <Expression>;` The "**:=**" operator assigns a value or expression to a variable.	`<VAR_Name>`	The symbolic name of the variable is a free identifier. The variable can be: • Variable of an elementary data type • Array element • Element of structured variables (STRUCT data type) • Input parameter when the function is called • Variable of predefined technological structure types
	`:=`	Assignment operator symbol
	`<Expression>`	An expression represents a value that is calculated when the program is executed.

Syntax	Designation	
`<VAR_Name> := <Expression>;` The ":=" operator assigns a value or expression to a variable.	`;`	Semicolon – terminates each expression
`<VAR_OUTPUT> => <VAR_Name>;`	`<VAR_OUTPUT>`	Symbolic name of the output parameter of a function. Free identifier
	`=>`	Assignment operator symbol
	`<VAR_Name>`	Name of the tag to which the value of the output parameter of the function is to be assigned. It must be possible to convert the type of the output parameter to the type of this tag using implicit conversion.

Example

The following example shows how to use assignment operators:

MCL

```
//assignment arithmetical expression
elemVar := 3 * 3;
//assignment tag with the same data type
elemVar := elemVar1;
//assignment array elements
elem1 := array[i];
array_1 [2] := array_2[5];
array [j] := 14;
//assignment whole array with the same data type
array_1 := array_2;
//assignment STRUCT data type elements
struct1.elem1 := Var1;
struct1.elem1 := 20;
struct1.elem1 := struct2.elem1;
//assignment input parameters in function call
mySub( in := 1.0 );
//assignment Output parameters in function call
myFunc( in := 10.0, Out => myVar2);
//assignment variables of predefined technological structured types
$A1.DynamicDefaults.Acceleration := 10.0;
myAxis := $A1;
myAxis := NULL;
myAxis1 := myAxis2;
IPR_StatusWord := $IPR.StatusWord;
```

7.5.3 Combined value assignment operators (S7-1500T)

Description

MCL provides the option to use combined value assignment operators, that is, combined assignment with arithmetic operators.
In this case the assignment operation is performed after the corresponding arithmetic operation. The first operand of this operation is the tag in the left part of the operator, the second operand is the expression in the right part.

Syntax

The table shows the syntax for using this option:

Syntax	Description	

`<VAR_Name> += <Expression>;`
`<VAR_Name> -= <Expression>;`
`<VAR_Name> *= <Expression>;`
`<VAR_Name> /= <Expression>;`
These operators assign a value after the arithmetic operation is performed.
	`<VAR_Name>`	Symbolic name of the tag. Free identifier.
		The tag can be:
		• Tag of an elementary data type
		• Array element based on an elementary data type
		• Element of a structured tag, based on an elementary data type
		• Input parameters of the function when calling functions

	`+=`	Combined assignment operator symbol. Accordingly: <VAR_Name> = <VAR_Name> + <Expression>
	`-=`	Combined assignment operator symbol. Accordingly: <VAR_Name> = <VAR_Name> - <Expression>
	`*=`	Combined assignment operator symbol. Accordingly: <VAR_Name> = <VAR_Name> * <Expression>

Syntax	Description	
<VAR_Name> += <Expression>; <VAR_Name> -= <Expression>; <VAR_Name> *= <Expression>; <VAR_Name> /= <Expression>; These operators assign a value after the arithmetic operation is performed.	/=	Combined assignment operator symbol. Accordingly: <VAR_Name> = <VAR_Name> / <Expression>
	<Expression>	An expression represents a value that is calculated when the program is executed.
	;	Semicolon – terminates each expression

Examples

The following examples of combined assignment with arithmetic operators are based on the assumption that the value of the MyVar tag is 10:

MCL

```
// assignment combined with addition
MyVar += 5; //result MyVar = 15;
// assignment combined with subtraction
MyVar -= 5; //result MyVar = 5;
// assignment combined with multiplication
MyVar *= 5; //result MyVar = 50;
// assignment combined with division
MyVar /= 5; //result MyVar = 2;
// Examples with array and structured variables
struct1.elem1 *= 20; // Match with struct1.elem1 := struct1.elem1 * 20
array_1[2] /= array_2[5]; // Match with array_1[2] := array_1[2] / array_2[5]
```

7.5.4 Assignment rules (S7-1500T)

Description

The following rules apply to assigning:

- If the variable has a simple data type, value assignment is possible under the following conditions:
  - The result of expression and variable must have an identical data type.
  - The data type of the calculated expression can be implicitly converted to the data type of the variable, provided this is permitted for the data types involved.
- If the variable is a complex data type or a data type predefined by the system, the assigned expression must be of the same data type.
- If the variable is an Array, the expression must also be an Array with the same data type for the Arrayelement and identical field boundaries.

Specifying values for structures and arrays using aggregates

Values for structures and arrays can be specified using aggregates. This is permitted in the following cases:

- Initialization within the variable declaration
- Parameters in system functions
- Direct assignment to one variable of the same type

Specifying values for structures and arrays across aggregates in a different context results in a runtime error (for e.g. in expressions, such as parameters in functions, etc.).

Example

MCL

```
(* Definition of data type (from outside the main programm) *)
TYPE
  myType : STRUCT
    id : DINT;
    values : ARRAY [0..1] OF LREAL;
  END_STRUCT;
END_TYPE

PROGRAM main
VAR
  (* Initialization within the tag declaration *)
  myVar2 : myType := ( id := 1, values := [1.0, 2.0] );
  myPos1 : TO_Struct_Ipr_Position := ( x := 10.0, y := 20.0, z := 30.0,
                                       A := 0.0, B := 0.0, C := 0.0 );
  myPos2 : TO_Struct_Ipr_Position;
END_VAR
myPos2 := ( x := 20.0, y := 40.0, z := 60.0, A := 0.0, B := 0.0, C := 0.0 );
(* technology function, complete specification of all structure members *)
linAbs( (x := 100.0, y := 100.0, z := 0.0, A := 1.0, B := 0.0, C := 0.0) );
(* technology function, incomplete specification of all structure member *)
(* Default-value for missing members in relative motions *)
linRel( ( x := 100.0 ), trans := 1, blend := 2 );
linRel( ( y := 100.0, z := 50.0 ) );
...
END_PROGRAM
```

Multiple assignments

Multiple assignments are permitted in MCL:

MCL

```
myVar12 := myVar2 := myVar3 := 0.0;
```

7.5.5 Arithmetic operators (S7-1500T)

Description

An arithmetic expression is an expression formed using arithmetic operators. These expressions allow you to process numerical data types.

Syntax

The following table shows for each arithmetic operation:

- The operator
- The permitted data types of the operands
- The data type of the result

Instruction	Operator	Data type		
		First operand	Second operand	Result
Exponential	**	LREAL	LREAL	LREAL
Unary plus	+	ANY_NUM	(None)	ANY_NUM
Unary minus	-	ANY_NUM	(None)	ANY_NUM
Multiplication	*	ANY_NUM	ANY_NUM	ANY_NUM
Division	/	ANY_NUM	ANY_NUM[1)]	ANY_NUM
Modulo division	MOD	ANY_INT	ANY_INT[1)]	ANY_INT
Addition	+	ANY_NUM	ANY_NUM	ANY_NUM
Subtraction	-	ANY_NUM	ANY_NUM	ANY_NUM

1) The second operand must not be zero.
The abbreviations have the following meaning:
ANY_INT for data types DINT, UDINT
ANY_NUM for data types ANY_INT, LREAL

Rules

The order in which the operators are applied within a mathematical expression depends on their priority.

You can find more information on the priority of operators in the section "Priority of operators (Page 120)".

- The division operator ('/', 'MOD') assumes that the second operand is not zero.
- If one number is of type ANY_INT (integer) and the other is of type LREAL (long real number), the result will always be of type LREAL.

Example of arithmetic expressions with numbers

Assuming that i and j are integer tags with values 11 and -3 (of data type DINT, for example), the following examples show integer expressions and their associated values:

Expression	Value
i + j	8
i - j	14
i * j	-33
i MOD j	-2
i / j	-3

7.5.6 Relational operators (S7-1500T)

Description

A relational expression is an expression of the BOOL type formed with relational operators.

Syntax

Relational operators compare the values of two operands and return a BOOL value as a result. The result obtained in this way is a value that represents either the TRUE or FALSE attribute. The meaning of the relational operators is shown in the following table:

Operator	Meaning
>	First operand is greater than the second operand
<	First operand is less than the second operand
>=	First operand is greater than or equal to the second operand
<=	First operand is less than or equal to the second operand
=	First operand is equal to second operand
<>	First operand is not equal to the second operand

The result of the relational expression is:

- 1 (TRUE) if the comparison is true
- 0 (FALSE) if the comparison is not true

The following table shows permitted combinations of data types for the two operands and relational operators:

Data type		Permissible relational operators
First operand	**Second operand**	
ANY_NUM	ANY_NUM[1)]	<, >, <=, >=, =, <>
ANY_BIT	ANY_BIT	<, >, <=, >=, =, <>
ARRAY	ARRAY[2)]	=, <>
Structure (STRUCT)	Structure (STRUCT)[2)]	=, <>
Variable of technological, structured type - AXIS_OBJECT	or NULL	=, <>

1) The comparison is performed in the lowest common data type to which both operands can be implicitly converted.

2) Data type of the first operand.
The abbreviations have the following meaning:
ANY_BIT for data types BOOL, DWORD
ANY_INT for data types DINT, UDINT
ANY_NUM for data types ANY_INT, LREAL

Rules

The following rules must be observed when creating relational expressions:

- Operands should be enclosed in parentheses to ensure that the instruction in which the logical operation is performed is unique.
- Logical expressions can be joined according to the rules of Boolean logic to create queries such as:
  ```
  IF a < b AND b > c THEN
  ...
  END_IF
  ```

Variables or constants of type BOOL and relational expressions can be used in the same way as the expression.

Example

The following example demonstrates the use of relational operators:

MCL

```
IF A = 2 THEN
 ; //...
END_IF;

var_1 := B < C; // var_1 of BOOL data type

IF D < E OR var_2 THEN // var_2 of BOOL data type
 ...
END_IF;
// Comparison operator Equal for variables of predefined
//technological structured type - AXIS_OBJECT
IF myAxis = $A1 THEN
```

MCL

```
...
END_IF
IF myAxis = NULL THEN
...
END_IF
// Comparison operator Unequal for variables of predefined
//technological structured type - AXIS_OBJECT
IF myAxis <> NULL THEN
...
END_IF
```

7.5.7 Logical operators (S7-1500T)

Description

With the AND, &, XOR, and OR logical operators it is possible to join operands and expressions of the general data type ANY_BIT (BOOL or DWORD).

The logical operator NOT allows you to negate operands and expressions of data type ANY_BIT.

Syntax

The table contains information about the available operators:

Instruction	Operator	First operand	Second operand	Result
Negation	NOT	ANY_BIT	-	ANY_BIT
Conjunction	AND or &	ANY_BIT	ANY_BIT	ANY_BIT
Exclusive disjunction	XOR	ANY_BIT	ANY_BIT	ANY_BIT
Disjunction	OR	ANY_BIT	ANY_BIT	ANY_BIT

The abbreviations have the following meaning: ANY_BIT for data types BOOL, DWORD.

Rules

- Logical expression: Only operands of the BOOL data type. The operators have the effects on the operands specified in the following truth table. The result of a logical expression is 1 (TRUE) or 0 (FALSE).
- Bit-serial expression: Operands of data type DWORD. The operators have the effect on the operands given in the following truth table:

Operands		Result				
a	b	NOT a	NOT b	a AND b a & b	a XOR b	a OR b
0	0	1	1	0	0	0

The operation is carried out bitwise.

Operands		Result				
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	1

The operation is carried out bitwise.

Examples

The following examples demonstrate the use of logical operators:

Expression (n = 10)	Value
(n > 0) AND (n < 20)	TRUE
(n > 0) AND (n < 5)	FALSE
(n > 0) OR (n < 5)	TRUE
(n > 0) XOR (n < 20)	FALSE
NOT ((n > 0) AND (n < 20))	FALSE

Expression	Value
2#01010101 AND 2#11110000	2#01010000
2#01010101 OR 2#11110000	2#11110101
2#01010101 XOR 2#11110000	2#10100101
NOT 2#01010101	2#10101010

7.5.8 Element selection operators (S7-1500T)

Description

The "." operator is used to select an element of the structure. It can only be used for tags of the STRUCT data type.

Syntax

To access (write or read) a specific element of a structure, it is necessary to use the "." symbol. The parent element of the structured tag must be followed by the symbol "." After this symbol, the child element must follow the structure:

```
<VAR_StructName>.<Parent_name>.<Child_name> := <Value>;
```

Example

MCL

```
PROGRAM Main
 VAR
     myMotor : STRUCT
          motorType : UDINT;
          Current : LREAL;
          Voltage : LREAL;
     END_STRUCT;
 END_VAR

// element selection
myMotor.Current := 10.5;
myMotor.Voltage := 380.0;
...
END_PROGRAM
```

7.5.9 Priority of operators (S7-1500T)

Description

The interpretation instruction of an expression depends on the following:

- Priority of operators used
- Left-after-right rule
- Use of brackets

Rules

Expressions are processed according to certain rules:

- Operator execution corresponds to their priority (see table below)
- Operators with the same priority are executed from left to right
- A minus sign at the beginning of an identifier means multiplication by -1
- Parentheses override the operator priority instruction, meaning that the contents of the parentheses have the highest priority
- The number of opening parentheses must be equal to the number of closing parentheses

Valid operators in MCL

The following table shows the operators valid in MCL with their execution priority level ("1" represents the highest priority):

Operator symbol	Operator	Priority
.	Element selection operators	1
(Expression)	Bracket contents	1
**	Exponential	2
-	Unary minus	3
NOT	Negation	3
*	Multiplication	4
/	Division	4
MOD	Modulo (remainder of division)	4
+	Addition	5
-	Subtraction	5
<	Less than	6
≤	Less than or equal to	6
>	Greater than	6
≥	Greater than or equal to	6
=	Equal	7
< >	Not equal	7
AND or &	Logical AND	8
XOR	Exclusive OR	9
OR	Logical OR	10
: =	Assignment value or expression to tag. This operator assigns a value or expression to a tag.	11
=>	Assignment of tag to output parameter. This operator assigns a tag to the output parameter when the function is called.	11
+=	Combined value assignment. This operator assigns a value after performing an arithmetic operation.	11
-=		11
*=		11
/=		11

7.6 Control instructions (S7-1500T)

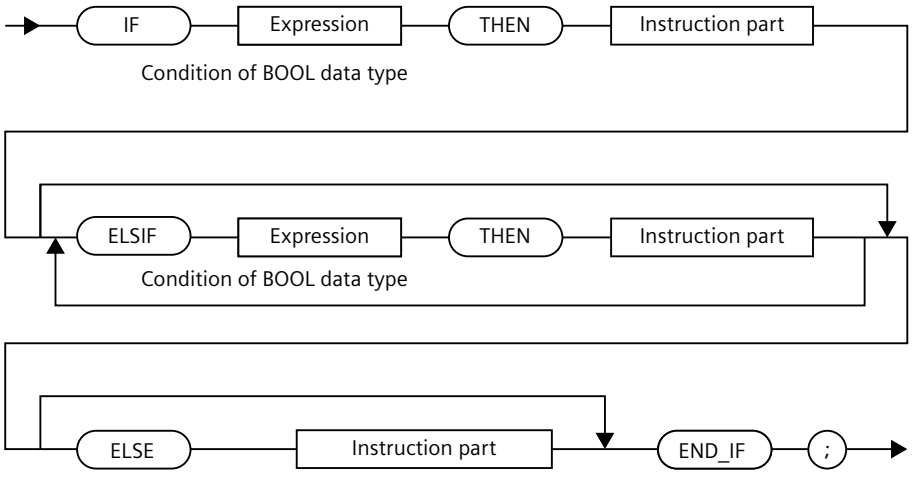
7.6.1 IF instructions (S7-1500T)

Description

The IF instruction is a conditional program branch. The execution sequence depends on the given conditions (one or more options). The execution of the conditional instructions forces the checking of the specified logical expressions. If the value of a logical expression is TRUE, the condition is met and the instruction part is executed. If the value is FALSE, the condition is not met and the instruction part is not executed.

Syntax

The following chart shows the syntax of an IF instruction:



Sequence of execution

The following applies to IF instructions:

- If the value of the first expression is TRUE, the instruction part is executed after THEN. The program is then continued after END_IF.
- If the value of the first expression is FALSE, the expressions in the ELSIF branches are then evaluated. If a Boolean expression in one of the ELSIF branches is TRUE, the instruction part following THEN is executed.
  The program is then continued after END_IF.
- If none of the Boolean expressions in the ELSIF branches are TRUE, the sequence of instructions following ELSE is executed (if there is no ELSE branch, no further instruction is executed).
  The program is then continued after END_IF.

Any number of ELSIF instructions can be programmed.

Note that ELSIF and ELSE instructions are optional.

One advantage of using one or more ELSIF branches instead of a sequence of IF instructions is that the logical expressions that follow valid expressions are no longer evaluated. This helps to reduce the processing time of the program and prevent the execution of unwanted program routines.

Example

The following example shows the use of the IF instruction:

MCL

```
(*
myAxis is a tag with type of AXIS_OBJECT.
It can be assigned to an object, using assign operator.
If the tag is not assigned to the AXIS_OBJECT (for example,
myAxis := $A1), the tag contains no AXIS_OBJECT and
has value NULL.
*)
IF myAxis <> NULL THEN //check if myAxis is assigned to AXIS_OBJECT
  myVar := 1;
ELSIF i = 4 THEN
  myVar := 2;
ELSE
  myVar := 3;
END_IF;
```

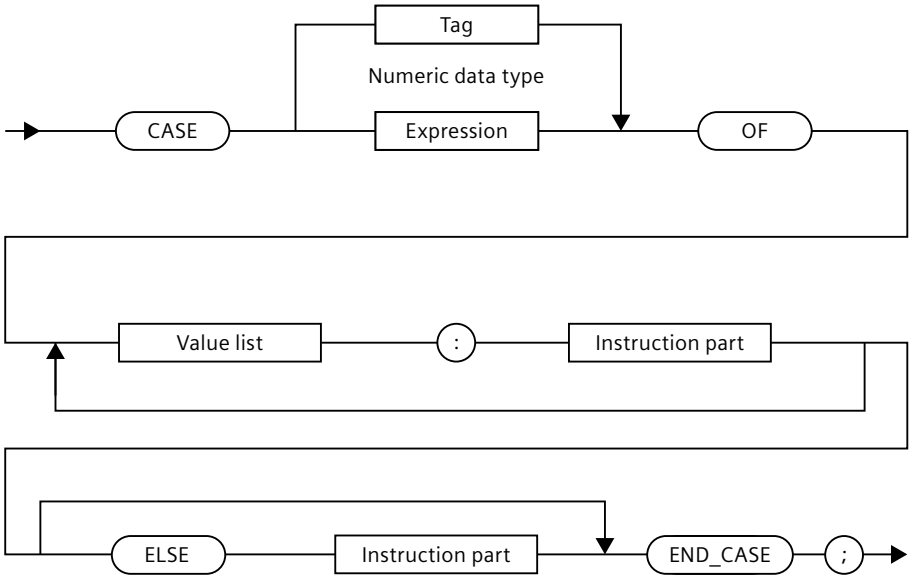
7.6.2 CASE instruction (S7-1500T)

Description

The CASE instruction selects a program section from a selection of n alternatives. This selection is based on the current value of a selection expression.
The selection is made from a list of values (value list) that assigns a program section to each value/group of values.

Syntax

The following chart shows the syntax of a CASE instruction:



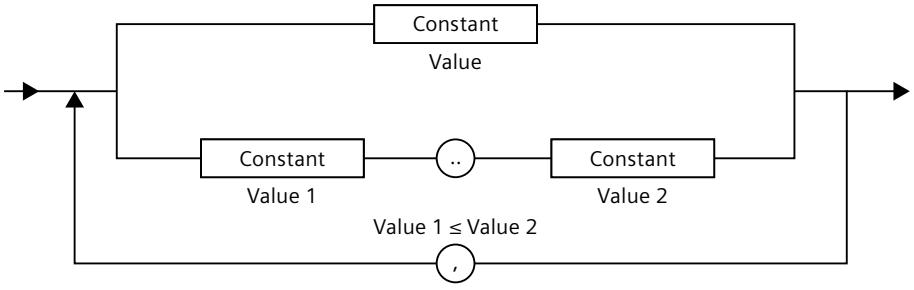
Sequence of execution

The following applies to CASE instructions:

- The expression following the CASE keyword is evaluated and must contain a value of type DINT or UDINT which is compared with the value list.
- It is then checked whether the selection value is listed in the value list. Each value in the list represents one of the permitted values for the selection expression.
- If there is a match, the corresponding part of the program is executed and the program then continues after END_CASE.
- If there is no match but an ELSE branch, this is executed. Otherwise, the program is continued after END_CASE.

Value list

The value list contains the permitted values for the selection expression. The following chart shows possible variants of the formulation of a value list:



Please note the following when formulating a value list:

- Each value list can begin with a constant (value), a list of constants (value1, value2, value3, ...) or a constant sequence (value1 to value2 - value1.. value2).
- The values in a value list must be integer constants.
- The same value in a value list may appear in several value lists. However, only the first occurrence is taken into account.

Example

The following example shows the use of a CASE instruction:

MCL

```
CASE myDintVar OF
  -1: myVar1 := 0;
  1 : myVar1 := 1;
          myVar2 := 2 * myVar1;
  2,3,4 : myVar1 := 3;
  5..6 : myVar1 := 4;
ELSE
  myVar1 := 0;
END_CASE;
```

In this example, the "myVar1" variable applies the value depending on the "myDintVar" value.

7.6.3 FOR instruction (S7-1500T)

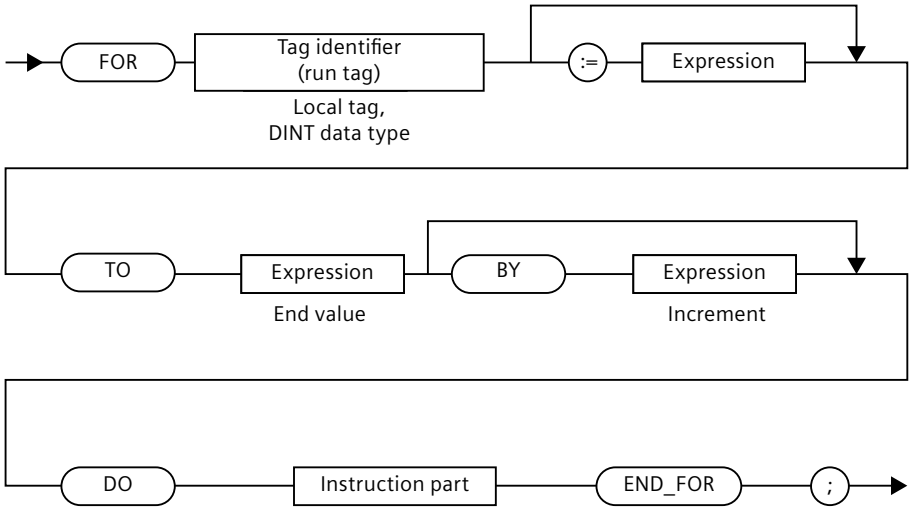
Description

As a repeat instruction, the FOR instruction executes a sequence of instructions in a loop and assigns values to tags on each run (run tag). The run tag must be a writable tag of type DINT. The FOR instruction is used when the number of executed loops in the programming phase is known.

The definition for a loop with FOR includes the specification of the start and end values of the run tag. Both start and end value expressions must be of the same data type as the run tag. If the number of runs is not known, the WHILE or REPEAT instruction is more suitable.

Syntax

The following chart shows the syntax of a FOR instruction:



Sequence of execution

The following applies to FOR instructions:

- At the beginning of the loop, the run tag is set to the start value and after each loop run through it is increased (positive increment) or decreased (negative increment) by the specified increment until the end value is reached.
- During each run it is checked whether the following conditions are met:
  - Value of the run tag ≤ end value (if the increment is positive) or
  - Value of the run tag ≥ end value (if the increment is negative).

  If the condition is met, the instruction part is executed. If the condition is not met, the loop and thus the instruction sequence is skipped and the program continues after END_FOR.
- If the loop cannot be executed due to the circumstances described in the previous point, the run tag retains its current value.

Rules

The following rules apply to the FOR instruction:

- The specification "BY" [increment] can be omitted. If no increment is specified, the default value is +1 (positive, end value ≥ start value) or -1 (negative, start value > end value).
- Start value, end value, and increment are expressions. The expression is evaluated once at the beginning of the FOR instruction.
- The run tag contains the value that causes the loop to terminate, which means that it is incremented before the loop ends.
- The run tag (current value) as well as the start value, end value, and increment must not be changed during the execution of the loop.
- It must be possible to convert the increment setting and the start and end values of the run tag to the type of the run tag using implicit conversion.

Example

The following example shows the use of a FOR instruction:

MCL

```
...
VAR
  i : DINT;
END_VAR
...
FOR i := 0 TO 10 BY 2 DO
    a := a + 1;
    // ...
    // command sequence
END_FOR;
```

In this example, the tag i increases by 2 on each run. If the value exceeds 10, the loop is terminated.

7.6.4 WHILE instruction (S7-1500T)

Description

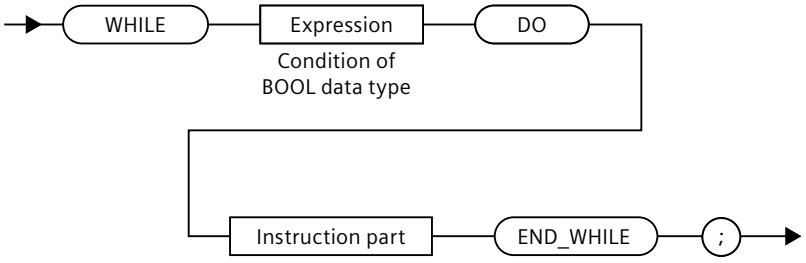
The WHILE instruction causes a sequence of instructions that are programmed between DO and END_WHILE. They are executed repeatedly if the condition specified in the instruction is met. The logical conditions are checked before each execution of the instruction sequence.

The instructions are repeated as long as the condition returns TRUE as a result of the test.

The WHILE instruction is advantageous when the number of executed loops is not known at the time of programming. If the number of executed loops is known, the FOR instruction is more suitable.

Syntax

The following chart shows the syntax of the WHILE instruction:



The instruction part after DO is repeated as long as the condition has the value TRUE.

Sequence of execution

The following applies to the WHILE instruction:

- The condition is always checked before executing the instruction part.
- If the value is TRUE , the instruction part is executed.
- If the value is FALSE, the WHILE instruction is terminated (this can occur during the first evaluation of the condition) and the program is continued after END_WHILE.

Example

The following example shows the use of a WHILE instruction:

MCL

```
...
VAR
  i : DINT := 1;
  isEnabled : BOOL := TRUE;
END_VAR
...
WHILE i <= 10 AND isEnabled DO
  // command sequence
  i := i + 1;
END_WHILE;
```

```
MCL
. . .
```

In this example, the loop is executed 10 times. The i variable can have the values 1 to 10.

7.6.5 REPEAT instruction (S7-1500T)

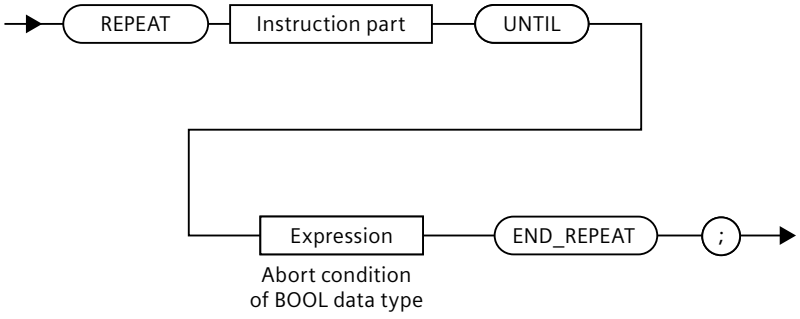
Description

The REPEAT instruction causes a sequence of instructions programmed between REPEAT and UNTIL to be executed repeatedly until the cancel condition is met. The cancel condition is formulated according to the rules for a logical expression.

The REPEAT instruction is advantageous when the number of executed loops in the programming phase is not known. If the number of executed loops is known, the FOR instruction is more suitable.

Syntax

The following chart shows the syntax of the REPEAT instruction:



Sequence of execution

The following applies to REPEAT instructions:

- The cancel condition is always checked after the instruction part has been executed. This means the instruction part is executed at least once, even if the cancel condition is TRUE at the start.
- If the value is FALSE, the instruction part is executed again.
- If the value is TRUE, the execution of the REPEAT instruction is terminated and the execution of the program resumes after END_REPEAT.

Example

The following example shows the use of a REPEAT instruction:

MCL

```
...
VAR
  i : DINT := 1;
  isEnabled : BOOL := TRUE;
END_VAR
...
REPEAT
  // command sequence
  i := i + 1;
UNTIL i = 10 AND isEnabled
END_REPEAT;
...
```

In this example, the loop is only executed 9 times because the i tag is initialized with 1 and incremented before the check. The tag can have the values 1 to 10.

7.6.6 EXIT instruction (S7-1500T)

Description

The EXIT instruction terminates a FOR-, WHILE- or REPEAT loop at any position independent of the conditions. The loop processing terminates immediately and the program resumes after END_FOR, END_WHILE or END_REPEAT. EXIT causes the cancellation of the loop that immediately surrounds the EXIT instruction.

Example

The following example shows the use of an EXIT instruction:

MCL

```
myIndex := 1;
FOR myIndex := 1 to 51 BY 2 DO
    IF myVar THEN
        EXIT;
    END_IF;
END_FOR;
(*
The following value assignment is performed after the execution of EXIT or
after the regular end of the FOR loop.
*)
Index_find := Index_2;
```

7.6.7 CONTINUE instruction (S7-1500T)

Description

The CONTINUE instruction terminates the current program run in a FOR-, WHILE- or REPEAT loop. After processing CONTINUE, the conditions for a continuation of the program loop (at WHILE and REPEAT) are queried, or the run variable is changed by the step size and checked to verify that it is still in the run range.

If the conditions are met, the next loop run begins after CONTINUE.

CONTINUE causes the cancellation of the program run of the loop that immediately follows the CONTINUE instruction.

Example

The following example shows the use of a CONTINUE instruction:

MCL

```
sum := 0;
FOR i := 1 TO 3 DO
    FOR j := 1 TO 2 DO
        sum := sum + 1;
        IF myVar THEN
            CONTINUE;
        END_IF;
        sum := sum + 1;
    END_FOR;
    sum := sum + 1;
END_FOR;
(*
Once the loop has run the "sum" tag has the following value:
With myVar = FALSE: sum = 15
With myVar = TRUE: sum = 9
*)
```

7.6.8 GOTO instruction (S7-1500T)

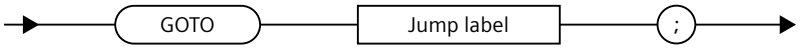
Description

The GOTO instruction is an unconditional instruction and causes a jump to a defined jump label. To use GOTO instructions it is necessary to program jump instructions with GOTO instructions and define jump labels.

GOTO instructions have to be used in certain circumstances, such as troubleshooting.

Syntax

The following chart shows the syntax of the GOTO instruction:



Jump label

To define a jump label , it is necessary to specify the symbolic name of the jump label before the instruction at which the program is to be continued.

The separation is made by a colon:

```
<Label_Name> : <Statement>;
```

Sequence of execution

The following applies to GOTO instructions:

- Only declared jump labels may be used.
- Jumps are only permitted within a function or the main program.
- Jumps ("up" or "down") to a label within the same control structure (IF, FOR, WHILE, REPEAT) are permitted.
- Jumps – up or down – to a label outside the current control structure (IF, FOR, WHILE, REPEAT) are also permitted. However, these may not occur in a new control structure that lies within.

Example

The following example shows the use of a GOTO instruction:

MCL

```
...
IF i > j THEN
  GOTO myLab1; //GOTO statement
ELSIF i > k THEN
  GOTO myLab2; //GOTO statement
END_IF;
...
myLab1: myVar := 1; // Jump label with statement
GOTO myLab3; //GOTO statement
myLab2 : myVar := 2; // Jump label with statement
...
myLab3 : myVar := 3; // Jump label with statement
...
FOR i := 1 TO 10 BY 1 DO
  ...
  IF i < 4 THEN
    GOTO myLab4; //GOTO statement
  END_IF;
  ...
  myLab4 : myVar := 4; // Jump label with statement
  ...
END_FOR;
```

7.6.9 RETURN instruction (S7-1500T)

Description

The RETURN instruction causes the currently executed function or the main program to be cancelled. When a function is terminated, the execution of the program continues in a parent program organization unit (POE) after the point from which the function was called.

Example

The following example shows the use of a RETURN instruction:

MCL

```
Index := 1;
FOR Index := 1 to 51 BY 2 DO
    IF myFlag THEN
        RETURN;
    END_IF;
END_FOR;
(*
The following value assignment is executed after the regular end of the FOR
loop, but not after the execution of RETURN.
*)
Index_find := Index_2;
```

7.6.10 SYNC instruction (S7-1500T)

7.6.10.1 Overview of the SYNC instruction (S7-1500T)

Description

You program the simultaneous execution of multiple instructions in a SYNC block.

The following two options are available to simultaneously execute instructions:

- Simultaneous start of instructions in the ON_START (Page 134)subblock
- Execution of synchronous actions (Page 141) in parallel with a path job:
  - Position-dependent synchronous actions in the ON_POS (Page 141) subblock

7.6.10.2 ON_START: Simultaneous start of instructions (S7-1500T)

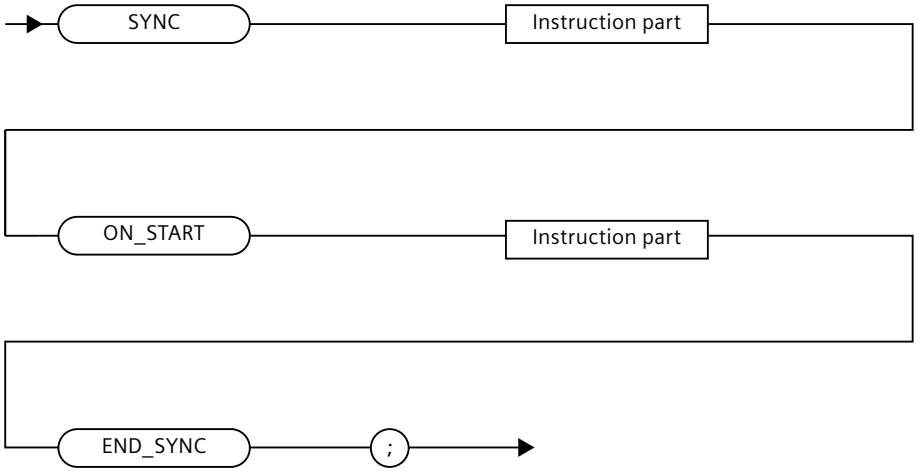
Description

In a ON_START subblock, you program the start of the execution of a MCL instruction simultaneously with another MCL instruction.

An ON_START block allows you to start an instruction at the blending point of a sequence instruction simultaneously with executing a sequence.

Syntax

The simultaneous start of the instruction is implemented in the SYNC/END_SYNC block. The sequences specified in the ON_START subblock start simultaneously with the main block:



MCL

```
SYNC
 // Main command or main sequence, with which further commands
 // are to be started synchronously
ON_START
 // commands to be started synchronously with the main sequence
END_SYNC;
```

If there are several instructions (sequences) in the SYNC or ON_START block, the instructions within the block are processed sequentially. Only the first instruction of the sequence in the ON_START block is started simultaneously with the first instruction of the sequence of the SYNC block.

Examples of using ON_START subblocks in a SYNC block can be found in section Examples of simultaneous start of instructions and instruction sequences (Page 136).

Rules

The following rules apply to the SYNC/END_SYNC block that contains one or more ON_START subblocks:

- The END_SYNC instruction is used as the synchronization point. Before the instructions following END_SYNC can be started, all instructions/instruction sequences started in the SYNC block must be completed according to their termination criterion. A programmed blending between instructions from the main sequence and after the END_SYNC instruction is ignored. The instructions are attached.
- Multiple ON_START subblocks are possible within a SYNC/END_SYNC block. The number of possible ON_START subblocks depends on the number of programmed instructions and the configuration of the Interpreter.SYNC or ON_START blocks that cannot be executed cause an alarm. You can change the number of prepared instructions in the program code (default value is 100) by adjusting the value of the following variable:

MCL

```
// set maximum depth of preprocessing queue for Interpreter to 50
$IPR.Parameter.MaxNumberOfCommands := 50;
```

- All ON_START subblocks within a SYNC/END_SYNC block must be fully preprocessed by the CPU in the preparation loop before execution, otherwise a runtime error occurs.
- Only the following instructions are permitted in the SYNC block and ON_START subblock:

Instruction	Description
`writeVar`	Write mapped PLC tag or technology object data block tag
`home`	Home technology object, set home position
`move`	Move axis with constant velocity
`posAbs`	Position axis absolutely
`posRel`	Position axis relatively
`linAbs`	Linear path motion with absolute position specification
`linRel`	Linear path motion with relative position specification
`circAbs`	Circular motion with absolute target position
`circRel`	Circular motion with relative target position
`ptpAbs`	sPTP motion with absolute target position in Cartesian coordinates
`ptpRel`	sPTP motion with relative target position in Cartesian coordinates
`ptpAxAbs`	sPTP motion with absolute target position in machine coordinates
`ptpAxRel`	sPTP motion with relative target position in machine coordinates
`ptpJtAbs`	sPTP motion with absolute target position in joint coordinates
`ptpJtRel`	sPTP motion with relative target position in joint coordinates
`trackIn`	Start conveyor tracking

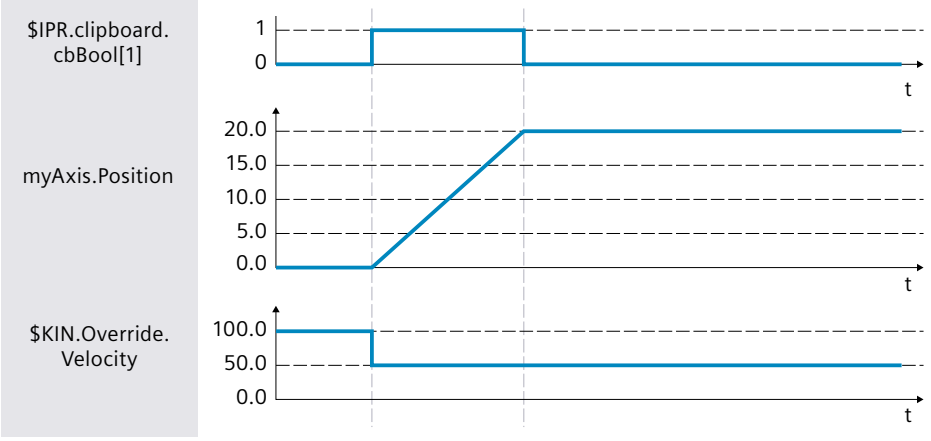
The use of other instructions is not permitted within this frame.

Examples of simultaneous start of instructions and instruction sequences (S7-1500T)

This section shows examples of simultaneous starts of instructions and instruction sequences.

Example 1 Simultaneous start of instructions

The following example shows the simultaneous start of the "posAbs" and "writeVar"" instructions with main instruction "writeVar"". The "writeVar" instruction is executed after the instructions are completed.

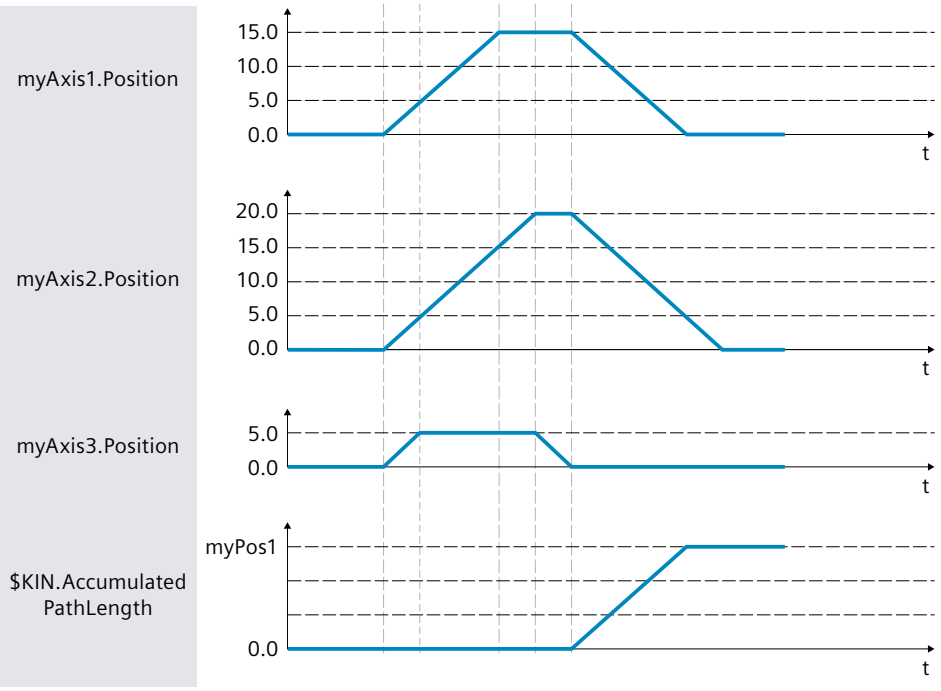


MCL

```
SYNC
  writeVar( $IPR.Clipboard.cbBool[1], TRUE );
ON_START
  posAbs( myAxis, 20.0 );
ON_START
  writeVar( $KIN.Override.Velocity, 50.0 );
END_SYNC;
writeVar( $IPR.Clipboard.cbBool[1], FALSE );
```

Example 2 Simultaneous start of instructions with two separate SYNC blocks

In the following example, the positioning of the axes myAxis1, myAxis2, and myAxis3 starts simultaneously (instruction "posAbs"). After all axes have reached their respective target position, myAxis3 is positioned to 0.0 (instruction "posAbs"). When this motion ends, a linear path motion occurs and the two axes (myAxis1 and myAxis2) are simultaneously positioned to 0.0.

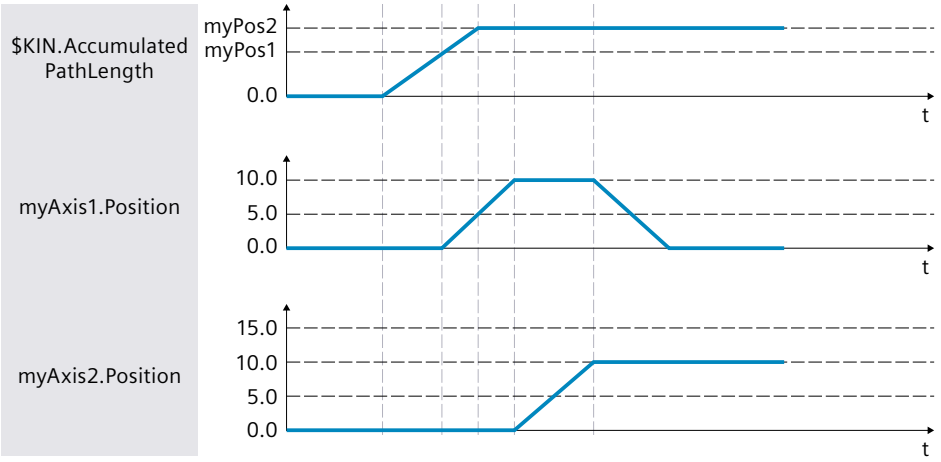


MCL

```
SYNC
  // Basic command to synchronize with
  posAbs( myAxis1, 15.0 )
ON_START
  posAbs( myAxis2, 20.0 );
ON_START
posAbs( myAxis3, 5.0 );
END_SYNC;
posAbs( myAxis3, 0.0 );
SYNC
  // Basic command to synchronize with
  linAbs( myPos1 );
ON_START
  posAbs( myAxis1, 0.0 );
ON_START
  posAbs( myAxis2, 0.0 );
END_SYNC;
```

Example 3 Simultaneous start at the blending point of a path motion

The simultaneous start syntactic construct (SYNC/END_SYNC) can be used to start parallel motion sequences at the blending point of the main sequence (e.g. path motion sequence with blending). The following example shows a simultaneous start at the blending point of the path motion. In the example, the positioning of myAxis1 to position 10.0 starts at the blending point of the path motion. After the positioning motion and path motion are completed, the positioning of myAxis2 to 10.0 and of myAxis1 to 0.0 follow sequentially.

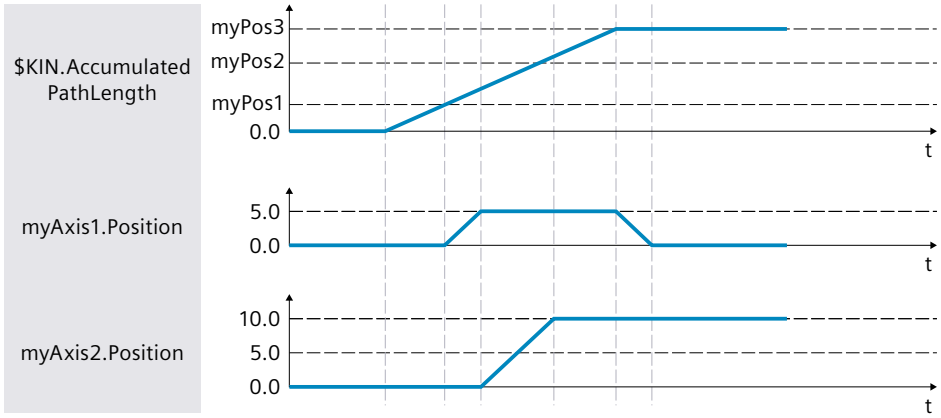


MCL

```
// active blending for TO_Kinematics
linAbs( myPos1, trans := 1, blend := 2 );
SYNC
// basic command to synchronize with
  linAbs( myPos2, trans := 0 );
ON_START
  // start in blending point
  posAbs( myAxis1, 10.0 );
END_SYNC;
posAbs( myAxis2, 10.0 );
posAbs( myAxis1, 0.0 );
```

Example 4 Simultaneous start of a sequence at the blending point of a path motion

In the following example, a sequence consisting of two sequential positioning instructions starts at the first blending point of the path motion. After completion of the path motion and the two positioning motions (myAxis1 to 5.0 and myAxis2 to 10.0), myAxis1 is positioned at 0.0.



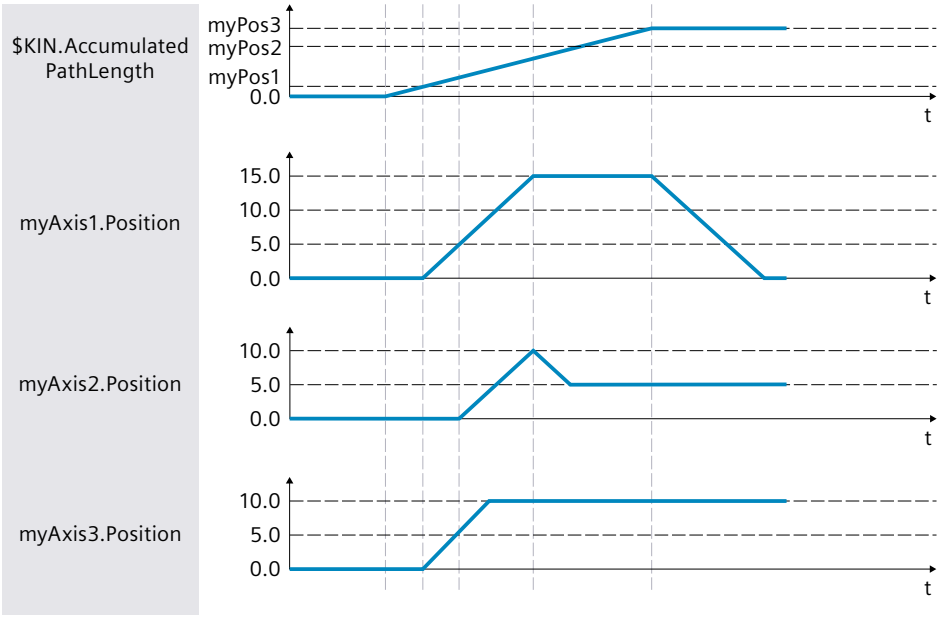
MCL

```
linAbs( myPos1, trans := 1, blend := 2 );
SYNC
  linAbs( myPos2, trans := 1, blend := 2 );
  linAbs( myPos3 );
ON_START
  // starts same time as first linAbs instruction in SYNC block
  posAbs( myAxis1, 5.0 );
  posAbs( myAxis2, 10.0 );
END_SYNC;
posAbs( myAxis1, 0.0 );
```

Example 5 Simultaneous start at the blending point of a path motion or a single axis motion sequence

It is possible to insert a SYNC/END_SYNC block into another SYNC/END_SYNC block. This enables instructions starting simultaneously to be cascaded.

In the following example, the positioning of the myAxis1 axis to 5.0 and simultaneously the positioning of the myAxis3 axis to 10.0 starts at the blending point of the path motion. The second SYNC block starts after positioning of myAxis1 is complete. The motion of myAxis1 is a single axis motion to position 15.0. Simultaneously with the "posAbs" instruction for myAxis1, another sequence starts, consisting of two sequential positioning instructions at the myAxis2 axis. myAxis1 is positioned to 0.0 at the end of all motions to be synchronized.



MCL

```
linAbs( myPos1, trans := 1, blend := 2 );
SYNC
  linAbs( myPos2, trans := 1, blend := 2 );
  linAbs( myPos3 );
ON_START
  // axial positioning
  posAbs( myAxis1, 5.0 );
  SYNC
    posAbs( myAxis1, 15.0 );
  ON_START
    posAbs( myAxis2, 10.0 );
    posAbs( myAxis2, 5.0 );
  END_SYNC;
ON_START
  posAbs( myAxis3, 10.0 );
END_SYNC;
  posAbs( myAxis1, 0.0 );
```

7.6.10.3 Synchronous actions (S7-1500T)

Description

A synchronous action is an instruction or sequence of instructions which is executed in parallel to a path job. A runtime event triggers the execution of a synchronous action.

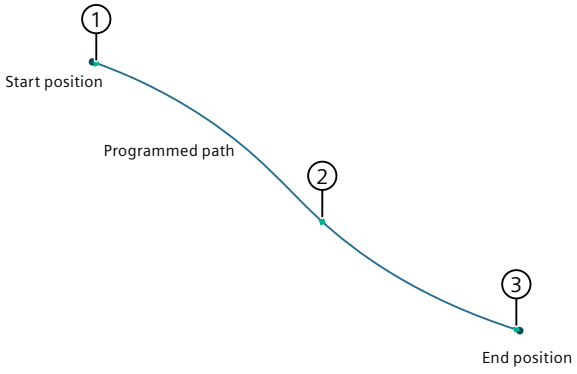
MCL supports position-dependent synchronous actions in the ON_POS (Page 141) subblock.

ON_POS: Position-dependent synchronous actions (S7-1500T)

Description

Position-dependent synchronous actions are executed in parallel with a path job.

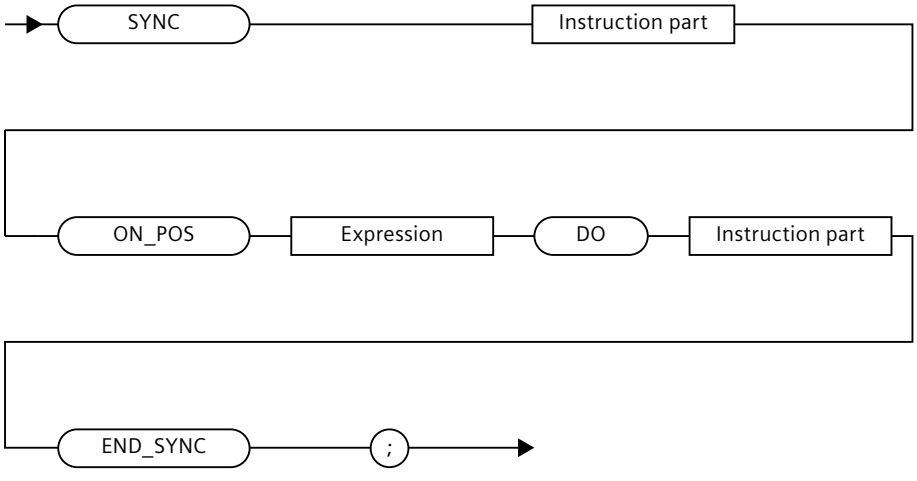
A position-dependent synchronous action is started at a specified path position.



① Synchronous action at the start position
② Synchronous action at a defined position
③ Synchronous action at the end position

Syntax

To program synchronous actions, you must use the keywords ON_POS and DO within the SYNC/END_SYNC block.



MCL

```
SYNC
  // main command or sequence to which the synchronous action refers
  // (e.g. path command or command sequence containing path commands only)
ON_POS [sType := <val>] [,p := <val>] [,t := <val>] DO
  // position triggered synchronous action
END_SYNC;
```

Examples of using ON_POS subblocks in a SYNC block can be found in section Examples of synchronous actions (Page 147).

Parameter

Parameter	Data type	Default value	Description	
sType	DINT	0	Type of position-dependent synchronous action (optional)	
			0	Start position Start of the synchronous action taking into account the start time at the beginning of the path sequence
			1	End position Start of the synchronous action taking into account the start time at the end of the path sequence
			2	Defined position Start of the synchronous action taking into account the start time at a defined path length of the sequence. "sType" := 2 is not supported for sPTP motions.
p	LREAL	0.0	Path position for "sType" := 2 (optional)	
t	time	T#0MS		Start time for "sType" := 0/1/2 (optional) The start time can also be negative (start time before reaching the path position)

You can find information on the parameter constellations in an ON_POS subblock in the section AUTOHOTSPOT.

Supported instructions in the ON_POS subblock

The entire main sequence (instructions after the keyword SYNC up to the first ON_POS block) is executed completely. The override mode can be specified on the synchronous action (ON_POS block). The synchronous action can include any sequence of the following instructions in the ON_POS block:

Instruction can be used in the ON_POS block of a synchronous action	Description
`setAxisDyn`	Set dynamic defaults for single-axis motions modally
`setAxisDynMax`	Set dynamic limits for single-axis motions modally
`setDyn`	Set dynamic defaults for path motions modally
`setDynMax`	Set dynamic limits for path motions modally
`setPlane`	Set main plane of circle path for circular path motions modally

Instruction can be used in the ON_POS block of a synchronous action	Description
`setCircDir`	Set orientation of circle path for circular path motions modally
`setDynAdapt`	Set dynamic adaptation for path motions modally
`setOriDyn`	Set dynamic defaults for orientation motions modally
`setOriDynMax`	Set dynamic limits for orientation motions modally
`setOriDirA`	Set direction of motion of Cartesian orientation A for linear, circular path and sPTP motions modally
`setPtpDyn`	Set dynamic defaults for sPTP motions modally
`setCs`	Set reference coordinate system for linear, circular path and sPTP motions modally
`setBlendDist`	Set blending distance for linear, circular path and sPTP motions modally
`setBlendFactor`	Set maximum blending distance for linear, circular path and sPTP motions modally
`setBlend`	Set blending mode for linear and circular path motions modally
`setTrans`	Set motion transition for linear, circular path and sPTP motions modally
`setLc`	Set target joint position space for sPTP motions modally
`setTurnJoint`	Set target joint position ranges for sPTP motions modally
`setOvr`	Set program override
`preHalt`	Halt program preparation
`writeVar`	Write mapped PLC tag or tag of technology object data block
`waitEvent`	Interrupt program execution until a specific event
`waitTime`	Interrupt program execution for a defined period
`setControlledByInterpreter`	Set "ControlledByInterpreter" bit for a technology object
`powerOn`	Enable axis
`powerOff`	Disable axis
`home`	Home axis, set reference point
`move`	Move axis with velocity/speed specification
`posRel`	Position axis relatively
`posAbs`	Position axis absolutely
`torqueLimitOn`	Activate force/torque limiting/fixed stop detection
`torqueLimitOff`	Deactivate force/torque limiting/fixed stop detection

Rules

- The start of a position-dependent synchronous action refers to the sequence of motions after the SYNC instruction.
- Only path motion jobs are permitted between SYNC and ON_POS blocks. For motions without blending, only one path job may be programmed. Blending may only be used to program the instruction of the path sequence.
- The path length of the motion job between the SYNC instruction and the subsequent ON_POS instruction serves as the trigger for the start of a position-triggered synchronous action (instructions after the keyword DO).
- The actions to be performed are programmed according to an ON_POS...DO instruction. Optionally, the behavior of the synchronous action can be controlled by block attributes ("sType", "p", "t") in the ON_POS instruction.
- The END_SYNC instruction represents the synchronization point with the further instructions in the Interpreter program. It means that further program execution will continue only when the synchronous actions are completed. A programmed blending between instructions from the main sequence and after the END_SYNC instruction is ignored. The instructions are attached.
- After the end of the synchronous action, the execution of the MCL program continues with the next instruction after the END_SYNC- instruction.
- If a parameterized path length is reached or exceeded (parameter p), the execution of the instructions in the synchronization block (instructions after the keyword DO), taking into account an optional start time ("t"), is triggered. The start time can be positive (start time after reaching the path position – see example 2) or negative (start time before reaching the path position – see example 3). If an instruction sequence consists of multiple consecutive instructions, a negative start time can result in a trigger point in the preceding instruction.
- The assignment of a path job or a path sequence (instructions with blending) can be made to different synchronous actions, i.e. different ON_POS sections. The maximum number of programmable synchronous actions is 10. Non-executable ON_POS blocks result in an alarm.
- The local and global tags declared, or made known, in the Interpreter program, are locked for the changes during synchronous actions. This means that the value that such an Interpreter tag has in the program preparation is also in effect during the synchronous action. These tags cannot be written in a synchronous action and a programmed assignment (assignment with operator ":=") is canceled with an error message. In the area between SYNC and the first ON_POS or ON_START command, path jobs are only permitted if at least one ON_POS block is programmed. This does not apply to the clipboard tags ($IPR.Clipboard.CbBool, $IPR.Clipboard.CbDint, $IPR.Clipboard.CbBool.CbLreal), technology object data block tags of all mapped axes, technology object data block tags of the Interpreter, technology object data block tags of the kinematics and mapped PLC tags. It follows that the values of the local and global tag declared in the Interpreter can be synchronized with the program execution by using the tags of the clipboard or the PLC in connection with a programmed stop in the preparation of the Interpreter program (while using the preHalt() instruction).
- The execution of a synchronous action can be terminated prematurely by using the EXIT_SYNC instruction. All instructions that are preprocessed up to the EXIT_SYNC instruction are executed. The synchronous action is then finished. Further program execution is continued after the corresponding END_SYNC block (see Example 6).

See also

Constellations of synchronous actions (Page 145)

Constellations of synchronous actions (S7-1500T)

The following constellations result from the programming of instructions in the SYNC block:

Constellation	Description

`SYNC`
`  linAbs( …, trans := 0 );`
`ON_POS sType := … ,p := … DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	Triggers the synchronous action by the instruction in line 2;
	Reference to path length in line 2.

`SYNC`
`  linAbs( …, trans := 0 );`
`  linAbs( …, trans := 0 );`
`ON_POS sType := … ,p := … DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	Triggers the synchronous action by the instruction sequence in lines 2 and 3 (without blending within the reference sequence);
	Reference to the (accumulated) path length of the sequence in lines 2 and 3.

`SYNC`
`  linAbs( …, trans := 1 );`
`  linAbs( …, trans := 0 );`
`ON_POS sType := … ,p := … DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	Triggers the synchronous action by the instruction sequence in lines 2 and 3 (with blending within the reference sequence);
	Reference to the (accumulated) path length of the sequence in lines 2 and 3.

`linAbs( …, trans := 1 );`
`SYNC`
`  linAbs( …,trans := 0 );`
`ON_POS sType := … ,p := … DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	Triggers the synchronous action by the instruction in line 3;
	Reference to path length in line 3.

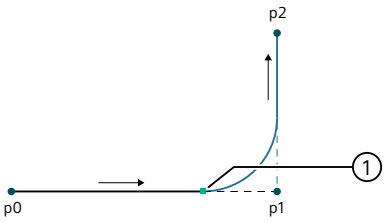
`linAbs( …, trans := 1 );`
`SYNC`
`  linAbs( …,trans := 0 );`
`  linAbs( …,trans := 0 );`
`ON_POS sType := … ,p := … DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	Triggers the synchronous action by the instruction sequence in lines 3 and 4 (without blending within the reference sequence);
	Reference to the (accumulated) path length of the sequence in lines 3 and 4.

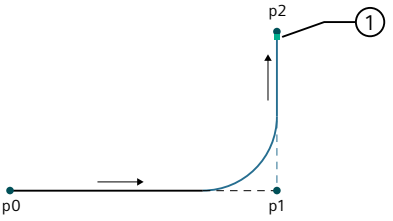
`linAbs( …, trans := 1 );`
`SYNC`
`  linAbs( …,trans := 1 );`
`  linAbs( …,trans := 0 );`
`ON_POS sType := … ,p := …`
`  // synchronous action`
`  …`
`END_SYNC;`
	Triggers the synchronous action by the instruction sequence in lines 3 and 4 (with blending within the reference sequence);
	Reference to the (accumulated) path length in lines 3 and 4.

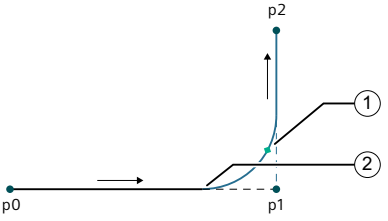
`SYNC`
`  linAbs( myPos1,... );`
`  writeVar( myVar, ... );`
`  linAbs( myPos2, ... );`
`ON_POS sType := ..., p := ...DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	For position-controlled synchronous actions, only path jobs are permitted. When using other instructions, such as "writeVar()", an error message appears.

Synchronous actions programmed in case of a blended path motion

The following table shows synchronous actions, programmed with a blended path motion and depending on the type of synchronous action ("sType" parameter):

`linAbs( p1, trans := 1, blend := 1 );`
`SYNC`
`  linAbs( p2, trans := 0 );`
`ON_POS sType := 0 DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	
	① Synchronous action at the start position

`linAbs( p1, trans := 1, blend := 1 );`
`SYNC`
`  linAbs( p2, trans := 0 );`
`ON_POS sType := 1 DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	
	① Synchronous action at the end position

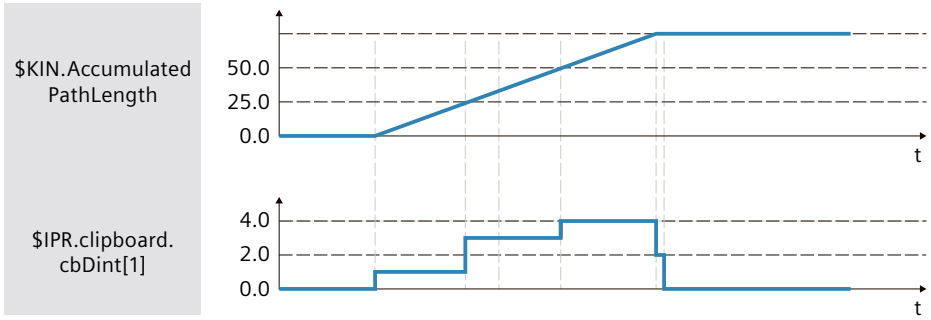
`linAbs( p1, trans := 1, blend := 1 );`
`SYNC`
`  linAbs( p2, trans := 0 );`
`ON_POS sType := 2, p := myLength DO`
`  // synchronous action`
`  …`
`END_SYNC;`
	
	① Synchronous action at a defined position (p = myLength)
	② p = 0.0

Examples of synchronous actions (S7-1500T)

This section shows examples of synchronous actions.

Example 1

In this example, position-triggered synchronous actions are programmed during a path motion with blending. Execution of the synchronous actions takes place as a function of the (accumulated) path length of the main sequence and the programmed trigger position ("sType" and "p"). In the synchronous actions, the Clipboard-Variable $IPR.clipboard.cbDint[1] is written to the corresponding lines/positions.



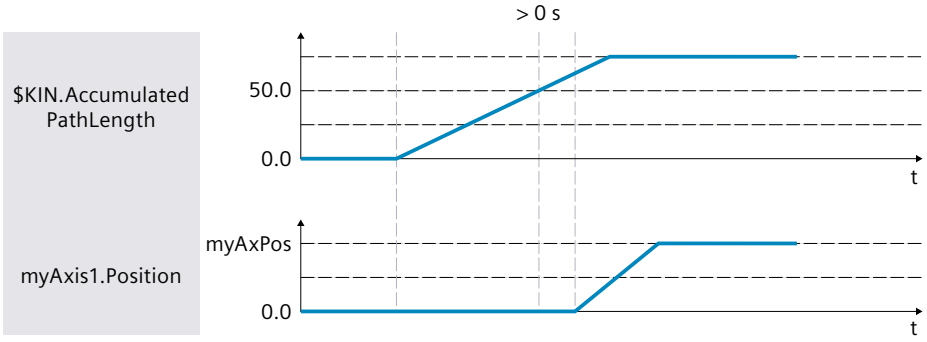
MCL

```
SYNC
  // path movement referred to
  linAbs( myPos1, trans := 1, blend := 2 );
  linAbs( myPos2, trans := 0 );
 ON_POS sType := 0 DO
  // execute at the beginning
  writeVar( $IPR.clipboard.cbDint[1], 1 );
 ON_POS sType := 1 DO
  // execute at the end
  writeVar( $IPR.clipboard.cbDint[1], 2 );
 ON_POS sType := 2, p := 25.0 DO
  //execute at path position 25.0
  writeVar( $IPR.clipboard.cbDint[1], 3 );
ON_POS sType := 2, p := 50.0 DO
  // execute at path position 50.0
  writeVar( $IPR.clipboard.cbDint[1], 4 );
END_SYNC;
writeVar( $IPR.clipboard.cbDint[1], 0 );
```

Example 2

In this example, the synchronous action starts 1 s later after the path position reaches 50.0. The myAxis1 axis positions itself after determining the position myAxPos.

The example shows the start of the synchronous action at a defined path length of a sequence ("sType" = 2), taking into account a positive start time.



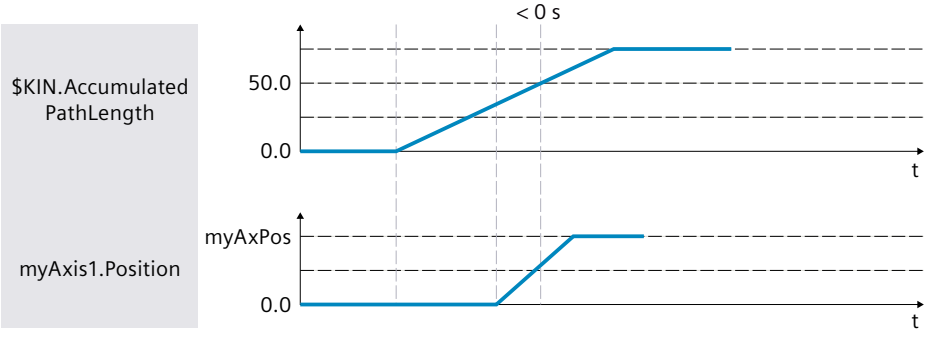
MCL

```
// determining the position using user-function
myAxPos := myFct();
SYNC
  linAbs( myPos );
 ON_POS sType := 2, p := 50.0, t := T#1s DO
  // myAxis1 is not kinematics axis
  posAbs( myAxis1, myAxPos );
END_SYNC;
```

Example 3

In this example, a negative start time is taken into account. The synchronous action starts 1 s (parameter "t") before the path position of the kinematics reaches 50.0 (parameter "p"). The myAxis1 axis positions itself after determining the position myAxPos.

The example shows the start of the synchronous action at a defined path length of the sequence ("sType" = 2), taking into account a negative start time.



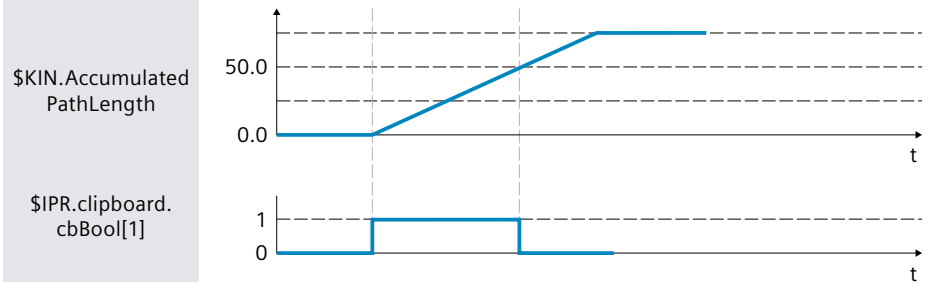
MCL

```
// determining the position using user-function
myAxPos := myFct();
SYNC
  linAbs( myPos );
 ON_POS sType := 2, p := 50.0, t := T#-1s DO
  // myAxis1 is not kinematics axis
  posAbs( myAxis1, myAxPos );
END_SYNC;
```

Example 4

In this example, the Clipboard Variable $IPR.clipboard.cbBool[1] of the Interpreter technology object is set to TRUE when the blended path motion starts. When the path length of the motion sequence reaches position 50.0, the same variable is set to FALSE.

The example shows how to use multiple synchronous actions that start with different conditions. The first action starts with the path start under the condition "sType" = 0. The second action starts after reaching the path position of the kinematics to 50.0 ("p"). In this example, the start time is 0 s ("t" = 0 as the default value).



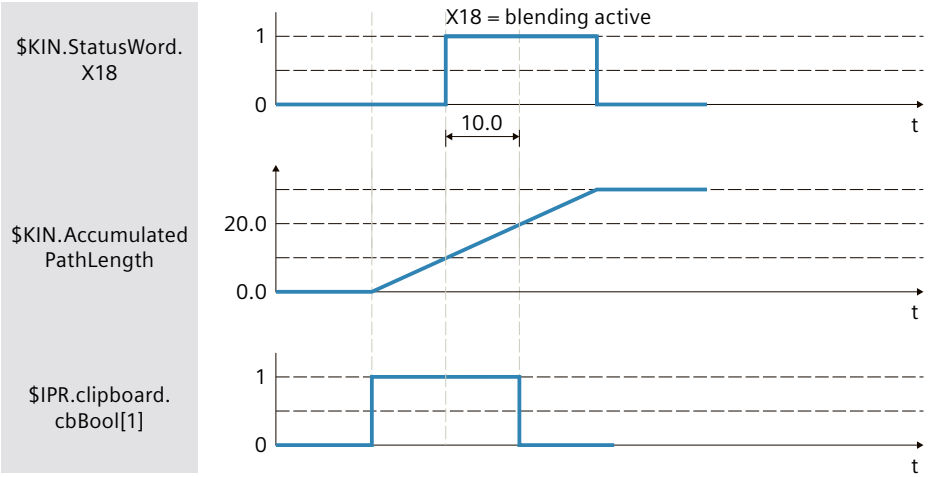
MCL

```
// 2 synchronous actions on a command sequence with blending
SYNC
  linAbs( myPos1, trans := 1, blend := 2 );
  linAbs( myPos2, trans := 0 );
 // with path start
 ON_POS sType := 0 DO
  writeVar( $IPR.Clipboard.cbBool[1], TRUE );
// execute at path position 55.0
 ON_POS sType := 2, p := 50.0 DO
  writeVar( $IPR.Clipboard.cbBool[1], FALSE );
END_SYNC;
```

Example 5

For cascading synchronous instructions, it is possible to insert a SYNC/END_SYNC block with a ON_POS subblock into another SYNC/END_SYNC block.

In this example, the Clipboard Variable $IPR.clipboard.cbBool[1] of the Interpreter technology object is set to TRUE when the blended path motion starts. When the path length reaches 10.0 ("p") of the path job myPos2, the same variable is set to FALSE.



MCL

```
// synchronous actions at a blended command sequence
// based on the path length of the single commands)
SYNC
  linAbs( myPos1, trans := 1, blend := 2 );
  SYNC
    linAbs( myPos2, trans := 0 );
  ON_POS sType := 2, p := 10.0 DO // on position 10.0
    writeVar( $IPR.Clipboard.cbBool[1], FALSE );
  END_SYNC;
ON_POS sType := 0 DO // with start of path to myPos1
    writeVar( $IPR.Clipboard.cbBool[1], TRUE );
END_SYNC;
```

Example 6

In this example, the instruction sequence in the synchronous action starts with the beginning of the path motion. Depending on the value of $IPR.clipboard.cbBool[1], further preparation and execution of the synchronous action ends prematurely with the EXIT_SYNC instruction.

MCL

```
SYNC
  // main command sequence
  linAbs( pos1 ); //
 ON_POS sType := 0 DO
 // command sequence
 FOR i := 1 TO 10 BY 5 DO
  ...
  // synchronous action prematurely ended
  IF $IPR.Clipboard.cbBool[1] = TRUE THEN
    EXIT_SYNC;
  END_IF;
 END_FOR;
```

MCL

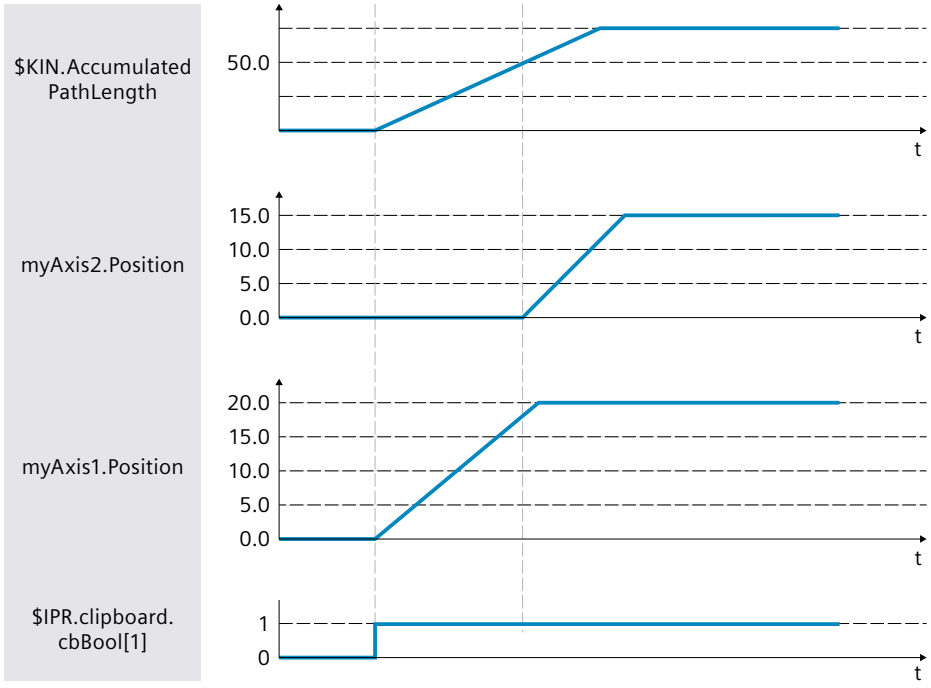
```
END_SYNC;
```

7.6.10.4 Combination of synchronous start and synchronous actions (S7-1500T)

Synchronous instructions or instruction sequences are programmed by SYNC/END_SYNC blocks. Within this unified block structure, specific subblocks can be used for the simultaneous start (ON_START subblock) and synchronous actions (ON_POS subblock). All synchronous actions programmed in the SYNC block refer to the same main sequence.

Example

In the following example, all synchronous actions refer to the same main sequence. The instructions in the ON_START blocks start simultaneously with the execution of the main sequence. The execution of the synchronous action starts at position 50.0 after the start of the execution of the main sequence.



MCL

```
SYNC
// main sequence
  linAbs( myPos1, trans := 1, blend := 2 );
  linAbs( myPos2, trans := 0 );
  ON_START
    // synchronous start with main sequence
    // myAxis1 is not kinematics axis
    posAbs( myAxis1, 20.0 );
  ON_START
    // synchronous start with main sequence
    writeVar( $IPR.Clipboard.cbBool[1], TRUE );
  ON_POS sType := 2, p := 50.0 DO
    //synchronous action at position 50
    // myAxis2 is not kinematics axis
    posAbs( myAxis2, 15.0 );
END_SYNC;
```

See also

preHalt() Stop program preparation (Page 328)
waitEvent() Interrupt program execution until a specific event (Page 318)

7.7 Structuring the MCL program (S7-1500T)

7.7.1 Overview of the MCL program structuring (S7-1500T)

Introduction

MCL programs must adhere to fixed structures and syntax rules. MCL consists of various program organization units (POEs) and provides options for program structuring and program organization. The MCL program to be processed by the Interpreter is described textually and supports the following structuring:

- Main program
- Functions
- User-defined data types

In addition to the main program, functions are the most important elements for program organization. In contrast to the main program, the programming of functions is optional.

The following figure shows a typical MCL program structure:

General POE structure

The source code for the main program and functions consists of the following sections:
- Start (mandatory)
- Declaration part (optional)
- Instruction part (mandatory)
- End (mandatory)

Start and end of a POE

Depending on the type of POE, the source code for a single POE is introduced by a default identifier for the start of the block and the block name. Completion is with a default identifier for the end of the block.

Name of the POE	Use	Syntax in MCL	Description

Main program	Obligatory	`PROGRAM Main`
		`.....`
		`END_PROGRAM`
			Entry point for running an Interpreter program.

Function	Optional	`FUNCTION <Name> : <Type>`
		`.....`
		`END_FUNCTION`
		`or`
		`FUNCTION <Name> : VOID`
		`.....`
		`END_FUNCTION`
			This POE can contain sequences with instructions. This POE must be programmed outside the main program:

User-defined data type	Optional	`TYPE`
		`.....`
		`END_TYPE`
			Used to declare new data types based on existing data types. This POE must be programmed outside the main program:

The order of the organization units of a program for declaring data types and functions is arbitrary, i.e. the "TYPE-END_TYPE" block (for user-defined data types) or "FUNCTION <Name> : <TYPE> - END_FUNCTION" (for functions) can be programmed after the first use.

Declaration part

The declaration part is used to define variables, constants, and parameters. A declaration part is divided into several declaration subsections, each separated by its own keyword pair. Each declaration subsection can appear only once in a POE. Not every declaration unit type is permitted in every POE type. There is no fixed sequence in which the declaration sections must be arranged.

The following table shows the declaration units and whether they are permitted (✓) or not permitted in a POE.

Name of the declaration unit	Use	Syntax in MCL	POE	
			Main program	Function

Declaration unit for local static variables	Optional	`VAR`
		`.....`
		`END_VAR`
			✓	✓

Declaration unit for local constants	Optional	`VAR CONSTANT`
		`.....`
		`END_VAR`
			✓	✓

Declaration unit for local temporary variables	Optional	`VAR_TEMP`
		`.....`
		`END_VAR`
			✓	✓

Declaration unit for global variables of the Interpreter	Optional	`VAR_IPR`
		`.....`
		`END_VAR`
			✓	

Declaration unit for global constants of the Interpreter	Optional	`VAR_IPR CONSTANT`
		`.....`
		`END_VAR`
			✓	

Declaration unit for input parameters	Optional	`VAR_INPUT`
		`.....`
		`END_VAR`
				✓

Declaration unit for output parameters	Optional	`VAR_OUTPUT`
		`.....`
		`END_VAR`
				✓

Instruction part

One or more sequences can be programmed in the instruction part. This part may contain:

- Value assignments to assign the result of an expression or the value of another variable to a variable.
- Control instructions to repeat instructions or groups of instructions or to branch within a program.
- Function calls
- MCL instructions

See also

Data types (Page 78)

7.7.2 Main program (S7-1500T)

Description

The main program is the entry point for running an interpreter program. By specifying the main program, the sequences to be processed are available in a defined processing order. The specification of the main program is mandatory.

Structure

The main program consists of four parts:

- Start (mandatory) - keyword "PROGRAM Main"
- Declaration part (optional) of up to four blocks:
  - Declaration unit for local constants (optional)
  - Declaration unit for local static tags (optional)
  - Declaration unit for global constants of the interpreter (optional)
  - Declaration unit for global tags of the interpreter (optional)
- Instruction/execution part (mandatory)
- End (mandatory) - keyword "END_PROGRAM"

The declaration part of the main program must be programmed before the instruction/execution part.

Each declaration block type ("VAR CONSTANT", "VAR_IPR CONSTANT", "VAR", or "VAR_IPR") in the declaration part is permitted only once and ends with "END_VAR".

The following figure shows the typical structure of a main program:

Syntax

The syntactic identifier of the main program is a block with the keywords "PROGRAM Main" (main program start) and "END_PROGRAM" (main program end). This block is permitted once per MCL program. The main program cannot be used as a function.

Name of the POE	Use	Syntax in MCL	Description

Main program	Obligatory	PROGRAM Main
		// declaration part
		VAR CONSTANT
		.....
		END_VAR
		
		VAR
		.....
		END_VAR
		
		VAR_IPR
		.....
		END_VAR
		
		VAR_IPR CONSTANT
		.....
		END_VAR
		//statement part
		END_PROGRAM
			Entry point for running an Interpreter program.

Example

The following example shows a main program:

MCL

```
PROGRAM Main
// declaration part
 VAR CONSTANT
   myConst1 : DINT := 10;
 END_VAR
 VAR
   myVar1, myVar2 : LREAL;
 END_VAR
 VAR_IPR
   myGlobalVar : LREAL;
 END_VAR
 VAR_IPR CONSTANT
   myGlobalConst1 : LREAL := 3.14;
 END_VAR
// statement part
 myVar1 := 1.5;
 myVar2 := 2.3 + myConst1 + myGlobalConst1;
END_PROGRAM
```

7.7.3 Functions (S7-1500T)

Description

A function can be called several times at different points in a program. It contains code that is executed whenever the function is called.

Functions can be used, for example, for the following purposes:

- Math functions, that return a result dependent on input values.
- Interrupt routines that are called in the main program, e.g. to control a conveyor belt.

Structure

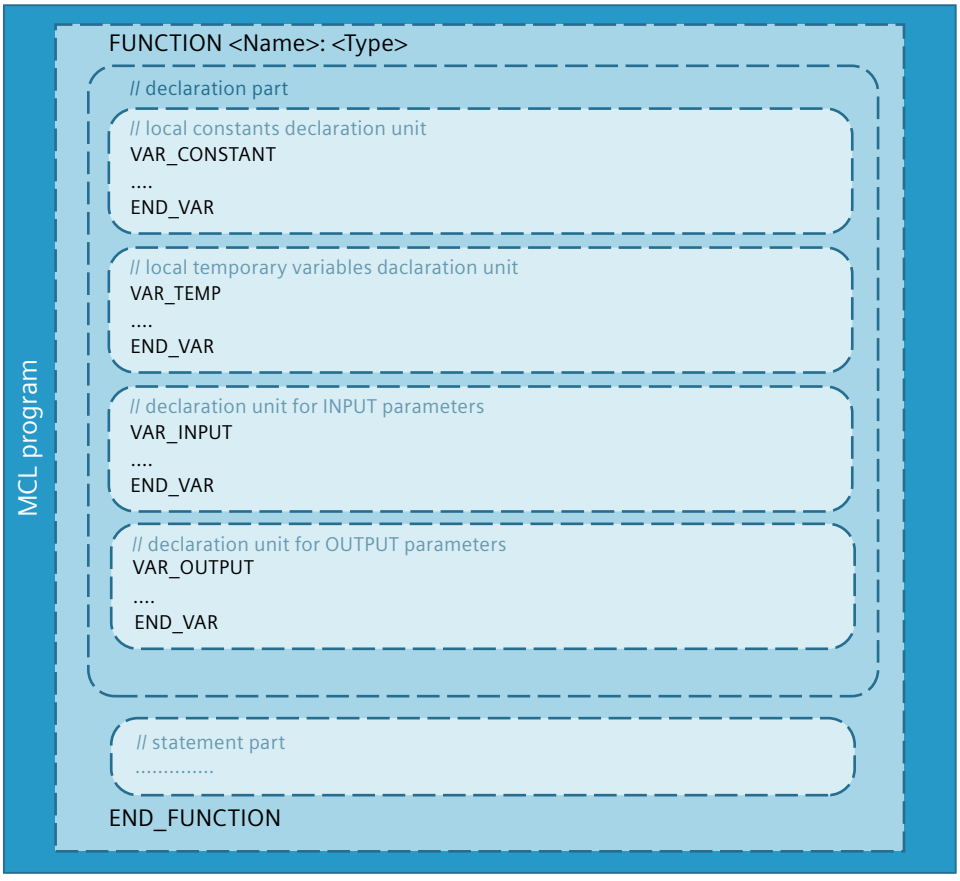
A function consists of two parts:

- Declaration part (optional) of up to five blocks:
  - Declaration unit for local temporary variables (optional)
  - Declaration unit for local constants (optional)
  - Declaration unit for input parameters (optional)
  - Declaration unit for output parameters (optional)
- Instruction/execution part (mandatory)

The arrangement and order of the units in the declaration part are arbitrary. The declaration part of a function must be programmed before the instruction/execution part. Each type of declaration unit (VAR_TEMP, VAR_INPUT, VAR_OUTPUT, VAR CONSTANT) is permitted only once in the declaration part, and ends with END_VAR.

Functions must be defined as an independent program unit before or after the main program. They can be used without a declaration part, regardless of the formulation of the function body before or after a function call.

The figure below shows the typical structure of a function.



Syntax

The syntactic identification of the function starts with the keyword FUNCTION (function start) and ends with the keyword END_FUNCTION (function end):

Syntax	Description	

`FUNCTION <Name> : <Type>`
`.....`
`END_FUNCTION`
	FUNCTION	This keyword marks the starting point of a function. The name is case sensitive.

	<Name>	The name of the function follows the keyword FUNCTION. The symbolic name of the function is a free identifier. The name is case sensitive.
	":"	Delimiter – separates the name of the function and the data type of the return value of the function.

Syntax	Description	

`FUNCTION <Name> : <Type>`
`.....`
`END_FUNCTION`
	<Type>	Specifies the data type of the return value.

	END_FUNCTION	This keyword terminates the function. The name is case sensitive.

`FUNCTION <Name> : VOID`
`.....`
`END_FUNCTION`
	The VOID type indicates that the function does not return a value as a result of its execution.	

Example

The following example shows the input parameters and output parameters that are declared in the corresponding declaration section of the function.

MCL

```
FUNCTION myFct: BOOL
 VAR_INPUT
     in1 : LREAL := 2.0;
     in2 : LREAL;
 END_VAR

 VAR_OUTPUT
     out : LREAL := 0.0;
 END_VAR

// Do something...
// return value
     out := SQRT( (in2 - in1)**2 );
     myFct := TRUE;
END_FUNCTION

// function without return value
FUNCTION mySub : VOID
 VAR_INPUT
    in : LREAL;
 END_VAR

 VAR_OUTPUT
    out : LREAL;
 END_VAR

// statement part
...
END_FUNCTION
```

7.7.4 Function parameters (S7-1500T)

Description

The calling interface of a function is determined by the definition of the variables within the function. This definition is made separately for input variables and output variables. This variable declaration specifies the name of variable via which the transferred value can be read (input variables) or changed (output variables) within the function.

When a function is called, the value is transferred using these variable names agreed for the function interface, which in this context can be referred to as formal parameter names. The input variables get their value by assigning the value of an expression to the formal parameter. The values provided in output parameters as a function result are transferred to another variable by means of output assignment via the formal parameter name.

Categories

Formal parameter	Declaration unit	Use	Description
Input	VAR_INPUT	Optional	Input parameters adopt the current input values when the function is called.
Output	VAR_OUTPUT	Optional	Output parameters transfer the current output values to the calling function. Data can be written to and read from them.

Defining parameters, local variables, and constants

All local tags and constants as well as formal parameters (input parameters, output parameters) must be defined in the declaration section of the function. Each declaration unit type is permitted only once.

Local data is only saved temporarily, as long as the function is active. For this reason, all formal input and output parameters defined in the declaration section of a function must be assigned with actual parameters as part of the function call.

To enable optional specification of parameters in function calls, MCL allows you to initialize input and output parameters with a default value in the declaration. If a default value is not specified, the default value of the data type is used.

If the default value is specified in the declaration, the following rules apply:

- For input parameters: The parameter is optional and can be omitted in the function call. If the parameter is not specified in the call, the default value programmed in the declaration is used.
- For output parameters: The default value programmed in the declaration is used.

If the default value is not specified in the declaration, the following rules apply:

- For input parameters: The parameter must be mandatorily assigned the current value in the function call.
- For output parameters: The default value of the data type is used.

You can find more information in the section "Data types (Page 78)".

The following applies to parameter transfer to functions:

Formal parameter	Elementary data type	Structured data type
Input	Copy[1)]	Copy[1)]
Output	Copy[1)]	Copy[1)]

1) When a function is called, the parameter value is copied into the function body. During execution, the value of the parameter remains unchanged.

7.7.5 Calling functions (S7-1500T)

Description

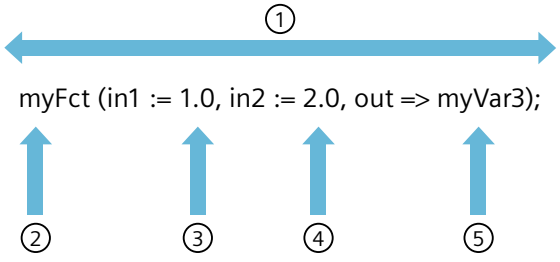
A function call is made by specifying:

- Function name (symbolic label).
- Parameter list with assignment of the required actual parameters in the instruction part of the main program or in the instruction part of another function.

Function call

The following example of a function call shows input parameters and output parameters.

Parameters are specified in the form of a value assignment. This value assignment assigns a value (actual parameter) to the parameters (formal parameters) defined in the declaration subsection of the called block.



① Parameter list
② Function name
③ Input parameter 1
④ Input parameter 2
⑤ Output parameter

Formal parameter	Transmission direction	Actual parameter
in1	←	1.0
in2	←	2.0
out	→	myVar3

When functions are called, data is exchanged between the calling block and the called block. The parameters to be transferred must be specified in the function call in the form of a parameter list. The parameters are enclosed in parentheses and separated by commas.

Parameter transfer

The following applies to parameter transfer:

- If the default values of input parameters are specified in the declaration part of the function, the specification of input parameters in function calls is optional.
- If the default values of input parameters are not specified in the declaration part, the current values must be assigned to the input parameters in the function call.
- Specification of output parameters in function calls is optional. If the default values of the output parameters are not specified in the declaration part of the function, the default value of the data type is used as the default value of the output parameter.
- If the data types do not match, an implicit type conversion of the actual parameters occurs. If this is not possible, a runtime error is issued.
- The assignment order is arbitrary.
- Individual assignments are separated by commas.

When using functions in expressions, the above conversion rules apply. If an expression does not expect an elementary data type, it must match the data type supplied by the function.

Example

The following example shows a function call:

MCL

```
// Call function "myFct"
// Parameter "in1" - optional (default value is specified in the declaration)
// Parameter "in2" - mandatory (default value is not specified in the declaration)
myVar1 := myFct( in1 := 1.0, in2 := 2.0, out => myVar3 );
myVar2 := myFct( in2 := 10.0 );
// Call function "mySub"
mySub( in := 1.0, out => myVar3 );
```

7.7.6 Return values of functions (S7-1500T)

Description

There are two ways to return the MCL function values:

- A function can be defined with or without a return value. This is specified via the return type in the function declaration. If the return type is specified with VOID, the function has no return value.
- Functions without return value cannot be used in assignment operations. Functions with return value can be used arbitrarily in expressions. The return value of the function is used equivalent to a variable to calculate the expression.

Example

The following example shows a return value in an expression:

MCL

```
//return value in an expression
(*result of functions FLOOR and myFct is written by assignment of the
function identifier to myVar*)

myVar := FLOOR( 123.4 + myFct( in1 := 2.0, in2 := 4.0 ) );
```

Return of arrays and structures

The return value of a function is available after it is called. All data types are permitted for the return value of functions. Return of arrays and structures is also supported. The same rules apply to handling return values in an expression as for variables of the same data type. If the return value of a function is a structure or array, it can be assigned to a variable of the corresponding data type.

Examples

The following example shows how to use the return values of the array or structure types:

MCL

```
FUNCTION myFctStruct : TO_Struct_Ipr_Position
//…
END_FUNCTION
FUNCTION myFctArray : ARRAY[0..1] OF LREAL
//…
END_FUNCTION
PROGRAM main
  VAR
    posVar: TO_Struct_Ipr_Position;
    arrayVar : ARRAY[0..1] OF LREAL;
  END_VAR

  posVar := myFctStruct();
```

MCL

```
  arrayVar := myFctArray();
END_PROGRAM
```

The return value is provided within the function implementation by assigning a value to the function name. This is true even if the return value is a structure or array. Individual components can then be accessed.

The following example shows how to set the return value of a function as a structure or array within the function implementation itself. The return value of a function can be assigned component-by-component for individual structure or array elements or by aggregation, i.e. the combination of several elements into a structured new element.

MCL

```
// example for a structure as return value
FUNCTION myFctStruct : TO_Struct_Ipr_Position
// by a tag value
myFctStruct := globalStruct;
// by single components
myFctStruct.x := 100.0;
myFctStruct.y := 100.0;
// using an aggregate:
myFctStruct := ( x := 100.0, y := 100.0, z := 100.0,
A := 100.0, B := 100.0, C := 100.0 );
END_FUNCTION
// example for an array as return value
FUNCTION myFctArray : ARRAY[0..1] OF LREAL
// by a variabel value

myFctArray := globArray;

// by single components
myFctArray [0] := 100.0;
myFctArray [1] := 101.0;

END_FUNCTION
```

7.7.7 Premature termination of functions (S7-1500T)

Description

The RETURN instruction terminates the current function at a location at which this instruction stands. After the function is terminated, program processing continues from the point from which the current function was called. You can find more information in the section "RETURN instruction (Page 133)".

Example

The following example shows how to terminate a function:

MCL

```
FUNCTION MySub : VOID
 VAR_TEMP
    i : DINT;
 END_VAR
// statement part
 //...
 IF i > 100 THEN
   RETURN;
 END_IF;
END_FUNCTION
```

7.7.8 Reserved key words (S7-1500T)

Description

To ensure compatibility of programs after possible extensions of MCL, unsupported program organization units are syntactically reserved.

The following program structuring options are syntactically reserved:

- Function blocks (FUNCTION_BLOCK)
- Organization blocks (ORGANIZATION_BLOCK)
- Data blocks (DATA_BLOCK)

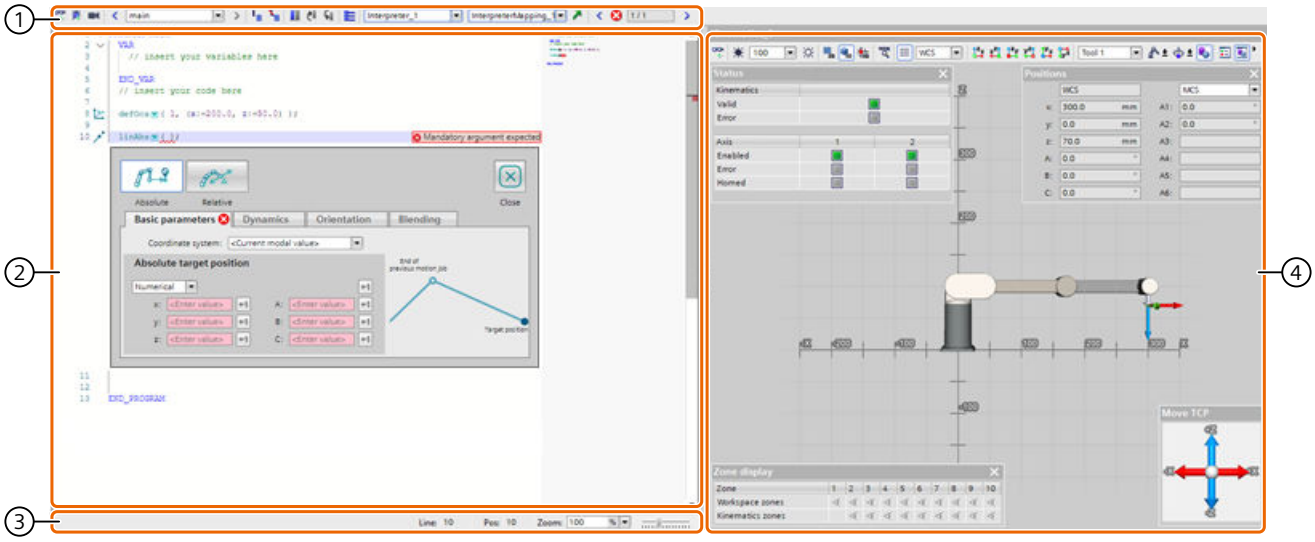
7.8 Working with the programming editor (S7-1500T)

7.8.1 Structure of the programming editor for the Interpreter programs (S7-1500T)

You can use the programming editor to create and edit Interpreter programs. The open Interpreter program can access the parameters of the Interpreter and Interpreter Mapping technology objects selected in the toolbar and the respective linked technology objects, e.g. kinematics.

Use the "Editor" entry in the project tree to open the programming editor of an Interpreter program technology object.

The following graphic shows the layout of the programming editor:

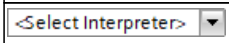
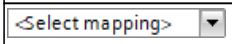
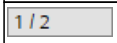


① Toolbar
② Programming window
③ Status bar
④ 3D visualization of the kinematics

Toolbar ①

The toolbar at the top of the programming editor provides you with the following functions:

Button	Function
	Monitoring On/Off
	Activate/deactivate Debug program mode For a description of the toolbar functions in Debug program mode, see the next table.
	Monitor program execution on/off
	Navigate to the previous function
main	Display the current function/Navigate to the selected function

Button	Function
	Navigate to the next function
	Comment
	Uncomment
	Set/remove bookmark
	Navigate to the previous bookmark
	Navigate to the next bookmark
	Closing all configuration dialogs
	Select technology object Interpreter
	Select technology object Interpreter Mapping
	Go to mapping configuration
	Navigate to the previous error
	Navigate to errors
	Show Number of the current error/Total number of errors
	Navigate to the next error

In Debug program mode, the toolbar offers the following additional functions:

Button	Function
	Continue program execution without a program preparation restart
	After change to tag values: Continue program execution with automatic restart of program preparation
	Continue program execution with a program preparation restart
	Cancel active motion instruction
	Stop program execution at the end of the running motion sequence
	Stop program execution at the end of the running MCL job
	Interrupt program execution at the next instruction
	Interrupt program execution at the end of the running motion sequence
	Step over without restart of program preparation
	After change to tag values: Step over with automatic restart of program preparation
	Step over with restart of program preparation
	Step in without program preparation restart
	After change to tag values: Step in with automatic restart of program preparation
	Step in with restart of program preparation

Button	Function
	Step out without program preparation restart
	After change to tag values: Step out with automatic restart of program preparation
	Step out with restart of program preparation
	Navigate to breakpoint

You can find more information on Debug program mode in section Testing Interpreter program (Page 56).

Programming window ②

You can create and edit an Interpreter program in the programming window.

The programming window is divided into the following areas:

- Left: Line numbering, code folding, quick actions, bookmarks, breakpoints
- Center: Programming area
- Right: Mini-preview of the programming area with colored markings

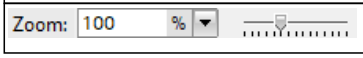
As programming support, "Configuration dialogs (Page 171)" are available in the programming window to facilitate parameter specification for instructions. In addition, the programming window contains functions for autocomplete, syntax checking, etc.

You can find more information about working with the programming window in the "Functions in the programming window of the programming editor (Page 169)" section.

You can find information on breakpoints in section Defining breakpoints for debugging (Page 63).

Status bar ③

The toolbar at the bottom of the programming editor provides the following functions:

Button	Function
	Shows line and position of the cursor
	Text size zoom

3D visualization of the kinematics ④

The 3D visualization contains a representation of the kinematics linked to a selected Interpreter technology object. To display the linked kinematics in the 3D visualization, select an Interpreter technology object in the "Select Interpreter" drop-down list in the programming editor toolbar.

You can find more information about working with 3D visualization of the kinematics technology object in the "3D visualization" section of the "S7-1500T Kinematics functions" (Page 10) documentation.

7.8.2 Functions in the programming window of the programming editor (S7-1500T)

You can create and edit an Interpreter program in the programming window of the programming editor. The following functions are available for this purpose:

- Quick actions
- Code folding
- Autocomplete
- Search and replace
- Syntax check
- Keyboard shortcuts

Quick actions

If there is no MCL instruction in a line yet, the symbol is displayed next to the line number. To insert a new MCL instruction in the line, click the icon and select the MCL instruction to be inserted from the menu.

If there is an MCL instruction in a line, the corresponding icon is displayed next to the line number. If you click on the corresponding icon of an MCL instruction, the menu with quick actions opens.

The following quick actions are available:

Icon	Quick actions	Description
	Insert	Inserts a new MCL instruction in the current empty line or below the current MCL instruction. Select the new MCL instruction from the menu.
	Duplicate	Duplicates the current MCL instruction. Any existing comments in the current line are not duplicated.
	Delete	Deletes the current MCL instruction. Any existing comments in the current line are not deleted.
	Change	Changes the current MCL instruction to a new MCL instruction. Select the new MCL instruction from the menu. Parameter values already stored in the current MCL instruction are transferred to the new MCL instruction as far as possible.

Code folding

If the ˅ symbol is displayed next to the line number, you can collapse the program section. Collapsible program sections begin with a keyword. It is also possible to collapse block comments.

Collapsed program sections can be recognized by the ˃ symbol next to the line number and by the blue marking of the line.

Autocomplete

You can activate autocomplete with the keyboard shortcut <Ctrl + spacebar>.

Autocomplete offers the following suggestions:

- Keywords
- MCL instructions
- Functions
- Control instructions
- Variables
- Data types
- Identifiers that are defined in declaration blocks
- Identifiers defined in the configuration of the Interpreter Mapping technology object selected in the toolbar.

Search and replace

You can find the "Search and Replace" function in the "Tasks" task card.

The following search options are available:

- Whole words only
- Match case
- Find in substructures
- Find in hidden texts
- Use wildcards
- Use regular expressions

The following replace options are available:

- Whole document
- From current position
- Selection

Syntax check

While editing the Interpreter program, a permanent syntax check takes place in the background. The syntax check detects syntax errors and semantic errors.

The detected errors and warnings are displayed in abbreviated form directly in the programming window. A list of all error messages with details is displayed in the Inspector window under "Info > Syntax".

You can use the list or the toolbar to navigate to the errors in the Interpreter program.

Keyboard shortcuts

The following keyboard shortcuts are available in the programming window:

Keyboard shortcut	Function
Ctrl + G	Go to line
Ctrl + Shift + W	Format selection
Ctrl + Shift + Y	Comment
Ctrl + Shift + U	Uncomment
Ctrl + spacebar	Show autocomplete list
Ctrl + Shift + M	Set/remove bookmark
Ctrl + Shift + 5	Navigate to the previous bookmark
Ctrl + Shift + 6	Navigate to the next bookmark
Ctrl + Shift + F2	Enable breakpoint
Ctrl + Shift + F3	Disable breakpoint
Ctrl + Shift + F7	Continue program execution
Ctrl + Shift + F9	Set/remove breakpoint
Ctrl + Shift + F10	Skip step
Ctrl + Shift + F11	Step in
Ctrl + Shift + F12	Step out

7.8.3 Working with the configuration dialogs of MCL instructions (S7-1500T)

"Configuration dialogs" are available to you for programming MCL instructions. The configuration dialogs support you in specifying parameter values. The configuration dialogs are displayed embedded in the programming window.

In the configuration dialogs, mandatory parameters are highlighted in red and optional parameters in orange until you have entered at least one value.

Opening a configuration dialog

If you use the fast actions (Page 169) to insert a new MCL instruction into the Interpreter program, the corresponding configuration dialog opens automatically.

To open a configuration dialog, follow these steps:

1. Click the icon next to the name of the MCL instruction.

Closing individual configuration dialogs

To close a configuration dialog, follow these steps:

1. Click the "Close" button in the configuration dialog. Alternatively, click the icon next to the name of the MCL instruction.

Closing all configuration dialogs

To close all configuration dialogs, follow these steps:

1. Click on the symbol in the toolbar.

Applying position values of the kinematics tool center point

When specifying coordinates, you have the option of applying the current position values of the tool center point (TCP) of the kinematics associated with the Interpreter selected in the toolbar. This function is only available if you have selected an Interpreter technology object in the toolbar that is linked to a Kinematics technology object.

To apply the coordinates of the tool center point, proceed as follows:

1. To apply the current position value of a coordinate, click the icon next to the corresponding input field.
2. To apply the current position values of all coordinates, click on the icon above the input fields.

7.8.4 Working with bookmarks (S7-1500T)

You can use bookmarks to mark program locations in extensive interpreter programs to make them easier to find for later revision. Bookmarks are displayed in the programming window in the left column next to the line numbers.

You can also use bookmarks in read-only mode.

Setting a bookmark

To set a bookmark, follow these steps:

1. Set the cursor in the appropriate line.
2. Click on the symbol in the toolbar of the programming editor. Alternatively, select the "Set bookmark" entry in the shortcut menu in the left column of the programming window.

Removing a single bookmark

To remove a bookmark, follow these steps:

1. Set the cursor in the appropriate line.
2. Click on the symbol in the toolbar of the programming editor. Alternatively, select the "Remove bookmark" entry in the shortcut menu in the left column of the programming window.

Removing all bookmarks

To remove all bookmarks, proceed as follows:

1. Select the "Remove all bookmarks" entry in the shortcut menu in the left column of the programming window.

Navigating to bookmarks

If a bookmark is in a collapsed program section, the bookmark is not displayed. If you navigate to a hidden bookmark, the program section is automatically expanded.

To navigate to bookmarks, follow these steps:

1. To navigate to the next bookmark, click on the icon in the toolbar of the programming editor.
2. To navigate to the previous bookmark, click on the icon in the toolbar of the programming editor.

7.9 Calling MCL instructions (S7-1500T)

7.9.1 Structure of a instruction call (S7-1500T)

The general structure of an MCL instruction consists of the function name followed by possible function parameters (in parentheses), separated by a comma.

There are three types of parameters for configuring an instruction in an MCL program:

- Required parameters
- Optional parameter with modal value
- Optional parameter with default value

Syntax

MCL

```
instrName( <nVal1>,…, <nValN> [,omPara1 := <val1>] [,…,omParaN := <valN>] );
```

Syntax element	Description
`instrName`	Symbolic function name
`()`	Instruction parameters (in brackets)
`nVal1…nValN`	List of values for required parameters. Only the value (in defined sequence) is specified for the required parameters
`omPara1…omParaN`	List of optional and/or modal parameters with parameter identifiers (parameter names)
`val1…valN`	Expression (value)

Syntax element	Description
:=	Assignment operator
,	Symbol for delimiting parameters in the parameter list
;	Function calls end with a semicolon

Rules

- The required parameters are at the top of the parameter list. Their sequence is fixed.
- These parameters can be followed by optional or modal parameters consisting of parameter identifier (parameter name), an assignment operator and an expression. The sequence of modal and optional parameters within the parameter list is variable.
- Some instructions may have no parameters. In these cases, the name is followed by parentheses and a semicolon at the end.

7.9.2 Parameter types (S7-1500T)

Each instruction consists of an instruction identifier and the following parameters. The parameters are specified by the particular instruction type.

The parameter sequence is only specified for necessary parameters. It is not specified for optional parameters.

Necessary parameters

Specifying parameters is required for the particular instruction. An example is the position specification for positioning instructions. Necessary parameters have no parameter identifier.

MCL

```
…
// "position_1" - necessary parameter;
// "trans" – optional parameter with modal value (modal parameter)
linAbs( position_1, trans := 0 );
// "$A1" and "20.0" - necessary parameters
posAbs( $A1, 20.0 );
…
```

Optional parameters with modal values (modal parameters)

Optional parameters with modal values are parameters that logically belong to instructions, but do not necessarily need to be specified at the MCL instruction call. Optional parameters with modal values are initialized with the default values of the kinematics objects and the axes when an MCL program is started. They retain their value until this is changed by a set instruction for modal parameters in the MCL program.

You can find more information in the section "Modal parameters (Page 179)"

Optional parameter with default values

Optional parameters with default values must be specified with the associated parameter identifier (parameter name) and the assignment operator.

If the parameter is not specified at the instruction, the default value is used.

MCL

```
…
home( $A1 ); // direct absolute homing to 0.0
                                    // use default values for non
                                    // specified optional parameters

home( $A2, mode := 1, p := 10.0 ); // direct relative homing to 10.0
                                    // with specification of optional
                                    // parameters "mode" and "p"
 …
```

7.9.3 Parameter transfer (S7-1500T)

There are several ways to transfer parameters when calling an instruction in the MCL program.

Using variables and values

You can use both variables and numeric values as parameters of MCL instructions. The data types of the variables must match the data type of the parameters:

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
    accelKin : LREAL := 5000.0;
  END_VAR

  (* using variables "Pos1", "accelKin" and numeric value for parameters
transfer *)
  linAbs( Pos1, v := 1000.0, a := accelKin );
END_PROGRAM
```

You can also use elements of arrays and structures as parameters of MCL instructions:

MCL

```
PROGRAM main
VAR
  Pos1 : TO_Struct_Ipr_Position;
  myArray : ARRAY [0..6] OF LREAL;
  ax : ARRAY [0..5] OF AXIS_OBJECT;
  myStruct : STRUCT
    v : LREAL := 2000.0;
    a : LREAL := 6000.0;
    d : LREAL := 6000.0;
    j : LREAL := 10000.0;
  END_STRUCT;
END_VAR
ax[2] := $A3;
myArray[1] := 1000.0;
myArray[2] := 5000.0;
// using elements of ARRAY for parameters transfer
linAbs( Pos1, v := myArray[1], a := myArray[2] );
posAbs( ax[2], 20.0 );
```

MCL

```
// using elements of STRUCT for parameters transfer
linAbs( Pos1, v := myStruct.v, j := myStruct.j );
END_PROGRAM
```

Using aggregates

You can use aggregates to specify the position in MCL instructions.

The following example shows how to use an aggregate for the "Target position" parameter with the "TO_Struct_Ipr_Position" data type:

MCL

```
// complete target position specification as aggregate
linAbs( ( x := 10.0, y := 20.0, z := 0.1,
          A := 25.0, B:= 45.0, C := 0.0 ) );
```

The sequence of the elements of the aggregate is undefined. It is not necessary to use all elements of the aggregate to specify the position in the MCL instruction.

Using an arithmetic expression

You can use an arithmetic expression as a parameter of MCL instructions. The result of the expression must match the data type of the parameter:

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
    myVar : LREAL;
  END_VAR

  // using arithmetic expression for parameter transfer
  linAbs( Pos1, v := 10 * myVar, j := 10000.0 );

  // using arithmetic expression for parameter transfer in aggregate
  linAbs( ( x := 100.0 + myVar, z := 25.0 + myVar * cos( 30.0 ) ) );
END_PROGRAM
```

Using literals

You can use system literals or variables accessed through system literals as parameters of MCL instructions.

The system literals "$A1", "$A2", "$A3", "$A4", "$A5", "$A6" for single-axis instructions can be used as "Axis instance" parameters:

MCL

```
// using literals for single axis commands
posAbs( $A1, 10.0 );
```

You can use the variables accessed through the system literals "$IPR" and "$KIN" as parameters in MCL command calls:

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
  END_VAR
  // using literal $IPR to transmit clipboard tag
  linAbs( Pos1, v := $IPR.Clipboard.CbLreal[1] );

  // using literal $KIN
  linAbs( Pos1, v := $KIN.DynamicDefaults.Path.Velocity );
END_PROGRAM
```

Using alias names from the mapping table.

As parameters in MCL command calls, you can use the alias names from the mapping table that are configured in the "InterpreterMapping" technology object ("TO_InterpreterMapping").

The following example shows the use of the alias name "myAxis" mapped to the technology object of a single axis (e.g. "TO_PositioningAxis") and the alias name "myPos" mapped to a LREAL variable of the global data block (see also sections "Mapping of variables (Page 41)" and "Mapping of technology objects (Page 40)"):

MCL

```
// using alias-names from mapping table
posAbs( myAxis, myPos );
```

7.9.4 Specifying absolute and relative target position (S7-1500T)

There are differences between absolute and relative motion commands when using default values for the "Target position" parameter.

Absolute positioning

If you do not use some elements of the aggregate in a kinematics motion job for absolute positioning, the job is executed without changing the coordinates not used in the call. The values of the missing coordinates are not changed to default values.

The last command of the following example is executed without changing the coordinate "x" to default values ("x" = 0).

MCL

```
PROGRAM main
  // position specification as aggregate in an absolute positioning command
  // target position is (10.0, 10.0, 10.0)
linAbs( ( x := 10.0, y := 10.0, z := 10.0 ) );

  // incomplete specification of aggregate members for target position
  // target position is (100.0, 10.0, 10.0), movement along x
linAbs( ( x := 100.0 ) );

  //target position is (100.0, 100.0, 10.0),
  //"x"-position remains unchanged, movement only along y
linAbs( ( y := 100.0 )
END_PROGRAM
```

For example, the kinematics only moves along the Cartesian y coordinate.

Relative positioning

If you do not use some elements of the aggregate in the kinematics motion job for relative positioning, the job is executed taking into account the default values for each unspecified coordinate. Since the default value of each coordinate is 0.0 ("x" = 0.0, "y" = 0.0, "z" = 0.0, "A" = 0.0, "B" = 0.0, "C" = 0.0), relative positioning is performed only in the specified coordinates.

MCL

```
PROGRAM main
  // target position is as before (unchanged), there is no relative move-
ment
  linRel( ( x := 0.0, y := 0.0, z := 0.0 ) );
  linRel( ( x := 100.0 ); // only x is changed relative to prev. position
  linRel( ( y := 100.0 ); // only y is changed relative to prev. position

END_PROGRAM
```

For example, the kinematics only moves along the Cartesian y coordinate.

7.10 Modal parameters (S7-1500T)

Modal parameters are initialized with the default values of the kinematics objects and the axes from the corresponding technology object data block when an MCL program is loaded and retain their value until it is changed by specific instructions in the MCL program.

For example, the start value for the velocity for the kinematics path motion is initialized from the "<TO_Kinematics>.DynamicDefaults.Path.Velocity" variable.

7.10.1 Modal parameters for single-axis instructions (S7-1500T)

The following table shows the modal parameters that act on the Axis technology object and the MCL instructions (set instructions of modal parameters) that are used to set them:

Modal dynamic value for MCL single instructions	Data type	Parameter	MCL instruction	Default value at MCL program start
Velocity	LREAL	v	`setAxisDyn` (Page 299)	"<TO_Axis>.DynamicDefaults.Velocity"
Acceleration	LREAL	a		"<TO_Axis>.DynamicDefaults.Acceleration"
Deceleration	LREAL	d		"<TO_Axis>.DynamicDefaults.Deceleration"
Jerk	LREAL	j		"<TO_Axis>.DynamicDefaults.Jerk"
Maximum velocity[1)]	LREAL	v	`setAxisDynMax` (Page 302)	"<TO_Axis>.DynamicLimits.MaxVelocity"
Maximum acceleration[1)]	LREAL	a		"<TO_Axis>.DynamicLimits.MaxAcceleration"
Maximum deceleration[1)]	LREAL	d		"<TO_Axis>.DynamicLimits.MaxDeceleration"
Maximum jerk[1)]	LREAL	j		"<TO_Axis>.DynamicLimits.MaxJerk"

[1)] Programmable only at the set instructions of modal parameters

7.10.2 Modal parameters for kinematics motions (S7-1500T)

The following tables show the modal parameters that act on the Kinematics technology object and the MCL instructions (set instructions of modal parameters) that are used to set them:

Path motion

Modal dynamic value for MCL kinematics motions	Data type	Parameter	MCL instruction	Default value at MCL program start
Velocity	LREAL	v	`setDyn` (Page 234)	"<TO_Kinematics>.DynamicDefaults.Path.Velocity"
Acceleration	LREAL	a		"<TO_Kinematics>.DynamicDefaults.Path.Acceleration"
Deceleration	LREAL	d		"<TO_Kinematics>.DynamicDefaults.Path.Deceleration"
Jerk	LREAL	j		"<TO_Kinematics>.DynamicDefaults.Path.Jerk"
Maximum velocity[1)]	LREAL	v	`setDynMax` (Page 242)	"<TO_Kinematics>.DynamicLimits.Path.Velocity"

[1)] Programmable only at the set instructions of modal parameters

Modal dynamic value for MCL kinematics motions	Data type	Parameter	MCL instruction	Default value at MCL program start
Maximum acceleration[1)]	LREAL	a	`setDynMax` (Page 242)	"<TO_Kinematics>.DynamicLimits.Path.Acceleration"
Maximum deceleration[1)]	LREAL	d		"<TO_Kinematics>.DynamicLimits.Path.Deceleration"
Maximum jerk[1)]	LREAL	j		"<TO_Kinematics>.DynamicLimits.Path.Jerk"
2D main plane	DINT	plane	`setPlane` (Page 258)	0 (XZ plane)
Circle path direction	DINT	cDir	`setCircDir` (Page 264)	0 (positive direction of rotation/shorter positive circle segment)
Mode of dynamic adaptation[1)]	DINT	da	`setDynAdapt` (Page 271)	"<TO_Kinematics>.DynamicDefaults.DynamicAdaption"

[1)] Programmable only at the set instructions of modal parameters

Orientation motion

Modal dynamic value for MCL kinematics motions	Data type	Parameter	MCL instruction	Default value at MCL program start
Velocity[1)]	LREAL	v	`setOriDyn` (Page 237)	"<TO_Kinematics>.DynamicDefaults.Orientation.Velocity"
Acceleration[1)]	LREAL	a		"<TO_Kinematics>.DynamicDefaults.Orientation.Acceleration"
Deceleration[1)]	LREAL	d		"<TO_Kinematics>.DynamicDefaults.Orientation.Deceleration"
Jerk[1)]	LREAL	j		"<TO_Kinematics>.DynamicDefaults.Orientation.Jerk"
Maximum velocity[1)]	LREAL	v	`setOriDynMax` (Page 245)	"<TO_Kinematics>.DynamicLimits.Orientation.Velocity"
Maximum acceleration[1)]	LREAL	a		"<TO_Kinematics>.DynamicLimits.Orientation.Acceleration"
Maximum deceleration[1)]	LREAL	d		"<TO_Kinematics>.DynamicLimits.Orientation.Deceleration"
Maximum jerk[1)]	LREAL	j		"<TO_Kinematics>.DynamicLimits.Orientation.Jerk"
Direction of orientation motion	DINT	oDirA	`setOriDirA` (Page 260)	3 (shortest path)

[1)] Programmable only at the set instructions of modal parameters

sPTP motion

Modal dynamic value for MCL kinematics motions	Data type	Parameter	MCL instruction	Default value at MCL program start
Velocity factor	LREAL	v	`setPtpDyn` (Page 240)	"<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor"
Acceleration factor	LREAL	a		"<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor"
Deceleration factor	LREAL	d		"<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor"
Jerk factor	LREAL	j		"<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor"
Target joint position space	DWORD	lc	`setLc` (Page 268)	16#FFFF_FFFF (Retain current link state)
Joint position area of joint 1	DINT	tj1	`setTurnJoint` (Page 262)	0 (shortest path)
Joint position area of joint 2	DINT	tj2		0 (shortest path)
Joint position area of joint 3	DINT	tj3		0 (shortest path)
Joint position area of joint 4	DINT	tj4		0 (shortest path)
Joint position area of joint 5	DINT	tj5		0 (shortest path)
Joint position area of joint 6	DINT	tj6		0 (shortest path)

Coordinate system

Modal dynamic value for MCL kinematics motions	Data type	Parameter	MCL instruction	Default value at MCL program start
Reference coordinate system	DINT	cs	`setCs` (Page 279)	0 (WCS)

Blending

Modal dynamic value for MCL kinematics motions	Data type	Parameter	MCL instruction	Default value at MCL program start
Motion transition	DINT	trans	`setTrans` (Page 247)	0 (append motion, no blending)
Blending mode	DINT	blend	`setBlend` (Page 250)	2 (Polynomial blending)
Blending distance	LREAL	blendDist	`setBlendDist` (Page 252)	-1.0 (maximum blending length of kinematics)
Blending factor[1)]	LREAL	blendFactor	`setBlendFactor` (Page 255)	"<TO_Kinematics>.Transition.FactorBlendingLength"

[1)] Programmable only at the set instructions of modal parameters

Program override

Modal dynamic value for MCL kinematics motions	Data type	Para-meter	MCL instruction	Default value at MCL program start
Program override[1)]	LREAL	over-ride	`setOvr` (Page 326)	"<TO_Kinematics>.Parameter.ProgramOver-ride"

1) Programmable only at the set instructions of modal parameters

7.10.3 Using modal parameters in MCL programs (S7-1500T)

Modal parameters are optional for MCL instructions.

If you use modal parameters in MCL instructions for motions, you need to specify the modal parameters with the appropriate parameter identifier and assignment operator before their value. If a modal parameter is specified directly in the MCL instruction for motion, the existing modal value of the modal parameter is not overwritten.

Only the MCL instruction in which the modal parameter is specified is executed with the specified value.

Example of using an instruction with modal dynamics parameters

In the following example, modal dynamics parameters (a, d, j) are specified with the set instruction "setDyn()". The velocity of the kinematics (v) is specified in the move instruction "linAbs()". The kinematics moves with the values of these dynamics parameters if these values are below the limits specified with "setDynMax()".

If no modal parameter is specified in the instruction, the last value defined by the specific MCL instruction is used.

MCL

```
PROGRAM main
 VAR
   Pos1 : TO_Struct_Ipr_Position;
 END_VAR

 // set max. values for dynamic
 setDynMax( v := 100.0, a := 10000.0, d := 10000.0, j := 10000.0 );

 // set dynamic parameters modally
 setDyn( a := 2000.0, d := 3000.0, j := 10000.0 );

 // linear movement of Kinematics to Pos1 with modal parameters (a, d, j),
 // specified with "setDyn()" and velocity specified in "linAbs()"
 linAbs( Pos1, v := 50.0 );

END_PROGRAM
```

Example of default modal dynamics parameters with a specific instruction

In the following example, the "linAbs()" instruction is used without specifying any modal dynamics parameters (only the required "Pos1" parameter is specified). The modal dynamics parameters are set with the special MCL instruction "setDyn()" before the MCL instruction "linAbs()".

The kinematics moves with the values set in "setDyn()".

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
  END_VAR

  // set dynamic parameters modally
  setDyn( v := 10.0, a := 2000.0, d := 3000.0, j := 10000.0 );
  // linear movement of Kinematics to Pos1 with modal parameters set
  // by instruction "setDyn()" before instruction "linAbs()"
  linAbs( Pos1 );

END_PROGRAM
```

If no modal parameter is specified in the MCL instruction and no special instruction was used to change it before calling, the default value is used as the value of this modal parameter.

Example of using initial values for modal dynamics parameters

In the following example, the "linAbs()" instruction is used without specifying the modal dynamics parameters (only the required "Pos1" parameter is specified).

Before this MCL instruction, no special MCL instructions are used that can change the value of the modal dynamics parameter. For example, if the start velocity value for the kinematics is 10.0 ("<TO_Kinematics>.DynamicDefaults.Path.Velocity" = 10.0), the kinematics will move at that velocity value ("v" = 10.0).

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
  END_VAR
  // movement of Kinematics to Pos1 with initial values of modal parameters
  linAbs( Pos1 );

END_PROGRAM
```

See also

Modal parameters for single-axis instructions (Page 179)

Modal parameters for kinematics motions (Page 179)

7.10.4 Limits of modal dynamics parameters (S7-1500T)

Limits of modal dynamics parameters are modal parameters. They are initialized at MCL program start and can be set with the following special MCL instructions:

- setAxisDynMax()
- setDynMax()
- setOriDynMax()

More information can be found in the sections "Modal parameters for single-axis instructions (Page 179)" and "Modal parameters for kinematics motions (Page 179)".

The values of the dynamics parameters in the MCL program are limited to the lowest of the following values:

- Value from the technology object data block:
  - Kinematics technology object "<TO_Kinematics>.DynamicLimits.*"
  - Axis technology object"<TO_Axis>.DynamicLimits.*"
- Modal value of the modal dynamics parameter set with a special MCL job
- Value of the modal dynamics parameter specified in the MCL job for the technology object motion.

Examples of limiting modal dynamics parameters

The following examples show limiting the velocity for the linear motion of the kinematics based on the value of the following variable in the technology object data block:

- "<TO_Kinematics>.DynamicLimits.Path.Velocity" = 100.0 mm/s.

The setting for velocity 100 mm/s is effective on the technology object data block and applies as the maximum value for path motions in the MCL program when loaded. Setting to 120.0 mm/s in "setDynMax()" or 110.0 mm/s in "linAbs()" does not cause any change, since the lower value from the technology object data block is still in effect.

The resulting velocity is 100 mm/s.

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
  END_VAR
  setDynMax( v := 120.0 ); // set limit for velocity
  linAbs( Pos1, v := 110.0 ); // linear movement with v = 100.0 mm/s

END_PROGRAM
```

The following example shows limiting the velocity for the linear motion of the kinematics based on the value set with a special MCL instruction ("setDynMax()").

The resulting velocity is 70.0 mm/s.

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
  END_VAR
  setDynMax( v := 70.0 ); // set limit for velocity
  linAbs( Pos1, v := 110.0 ); // linear movement with v = 70.0 mm/s

END_PROGRAM
```

The following example shows limiting the velocity for the linear motion of the kinematics based on the value set in the MCL instruction for the linear motion ("linAbs()").

The resulting velocity is 55.0 mm/s.

MCL

```
PROGRAM main
  VAR
    Pos1 : TO_Struct_Ipr_Position;
  END_VAR
  setDynMax( v := 70.0 ); // set limit for velocity
  linAbs( Pos1, v := 55.0 ); // linear movement with v = 55.0 mm/s

END_PROGRAM
```

MCL instructions (S7-1500T) 8

8.1 Kinematics motions (S7-1500T)

8.1.1 linAbs() Position kinematics absolutely with linear path motion (S7-1500T)

Description

With the MCL instruction "linAbs()", you move a kinematics with a linear motion to an absolute position.

The MCL instruction offers you the following options:

- Define absolute target position (depending on the kinematics, this means a Cartesian specification of the target position (x, y, z) and an orientation in space (A, B, C))
- Define dynamics
- Define motion transition
- Define blending mode

The MCL instruction "linAbs()" ends when the absolute end position is reached by the kinematics or blending is started by the next job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
linAbs( <pos> [,v := <value>] [,a := <value>] [,d := <value>] [,j := <value>]
[,trans := <value>] [,blend := <value>] [,blendDist := <value>] [,cs := <value>]
[,oDirA := <value>] );
```

Parameters

The following table shows the parameters of the instruction "linAbs()":

Parameter	Data type	Default value	Description	
pos	TO_Struct_Ipr_Position	-	Absolute target position in the specified reference coordinate system	
x	LREAL	0.0	x coordinate	
y	LREAL	0.0	y coordinate	
z	LREAL	0.0	z coordinate	
A	LREAL	0.0	A coordinate	
B	LREAL	0.0	B coordinate[1]	
C	LREAL	0.0	C coordinate[1]	
v	LREAL	-	Velocity[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Velocity". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
a	LREAL	-	Acceleration[2]	

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

Parameter	Data type	Default value	Description	
a	LREAL	-	-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Acceleration". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
d	LREAL	-	Deceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Deceleration". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
j	LREAL	-	Jerk[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Jerk". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			= 0.0	No jerk limit
			< 0.0	Not permitted
trans	DINT	-	Motion transition to the next job[2]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
blend	DINT	-	Blending mode[2]	
			-	The modal value is used. By default, the "blend" parameter is preset to the value "2". With setBlend() (Page 250), you can set another value modally.
			0	Blending in path dynamics (direct blending)
			1	Reserved
			2	Geometric blending with polynomial path
blendDist	LREAL	-	Blending distance[2]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

Parameter	Data type	Default value	Description	
blendDist	LREAL	-	≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.
cs	DINT	-	Reference coordinate system[2]	
			-	The modal value is used. By default, the "cs" parameter is preset to the value "0". With setCs() (Page 279), you can set another value modally.
			0	World coordinate system (WCS)
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
			3	Object coordinate system 2 (OCS3)
oDirA	DINT	-	Direction of movement of the Cartesian orientation A[2]. Only relevant for up to 4 interpolating kinematics axes.	
			-	The modal value is used. By default, the "oDirA" parameter is preset to the value "3". With setOriDirA() (Page 260), you can set another value modally.
			1	Positive direction
			2	Negative direction
			3	Shortest path If the target orientation can be reached by two paths of the same length, the motion is in positive direction.

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

Comparable Motion Control instructions

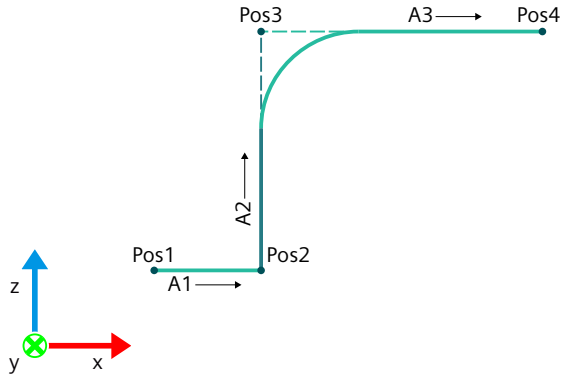
- "MC_MoveLinearAbsolute": Position kinematics with linear path motion

Example: Position kinematics with linear path motion

In the following example, a kinematics is to move to three positions.

The kinematics should move from standstill Pos1 to Pos2, and stop at Pos2 (A1).

From Pos2, the kinematics is to move with the blending distance "blendingDist" via Pos3 (A2) to Pos4 (A3).



MCL

```
VAR
   Pos2, Pos3, Pos4 : TO_Struct_Ipr_Position;
   blendingDist : LREAL;
END_VAR
// A1 - Exact stop in Pos2
linAbs( Pos2, trans := 0, cs := 0 );
// A2 - Blending high velocity in Pos3
linAbs( Pos3, trans := 2, blend := 2, blendDist := blendingDist, cs := 0 );
// A3 - Exact stop in Pos4
linAbs( Pos4, trans := 0, cs := 0 );
```

See also

Data type TO_Struct_Ipr_Position (Page 84)

8.1.2 linRel() Position kinematics relatively with linear path motion (S7-1500T)

Description

With the MCL instruction "linRel()", you move a kinematics with a linear motion relative to the position which was present at the start of the job processing.

The MCL instruction offers you the following options:

- Define relative target position (depending on the kinematics, this means a Cartesian specification of the target position (x, y, z) and an orientation in space (A, B, C))
- Define dynamics
- Define motion transition
- Define blending mode

The MCL instruction "linRel()" ends when the relative end position is reached by the kinematics or blending is started by the next job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
linRel( <pos> [,v := <value>] [,a := <value>] [,d := <value>] [,j := <value>]
[,trans := <value>] [,blend := <value>] [,blendDist := <value>] [,cs := <value>] );
```

Parameters

The following table shows the parameters of the "linRel()" instruction:

Parameter	Data type	Default value	Description
pos	TO_Struct_Ipr_Position	-	Relative target position in the specified reference coordinate system
x	LREAL	0.0	x coordinate
y	LREAL	0.0	y coordinate
z	LREAL	0.0	z coordinate

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

Parameter	Data type	Default value	Description	
A	LREAL	0.0	A coordinate	
B	LREAL	0.0	B coordinate[1]	
C	LREAL	0.0	C coordinate[1]	
v	LREAL	-	Velocity[2]	
		-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Velocity". With setDyn() (Page 234), you can set another value modally.
		> 0.0	The specified value is used.
		≤ 0.0	Not permitted
a	LREAL	-	Acceleration[2]	
		-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Acceleration". With setDyn() (Page 234), you can set another value modally.
		> 0.0	The specified value is used.
		≤ 0.0	Not permitted
d	LREAL	-	Deceleration[2]	
		-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Deceleration". With setDyn() (Page 234), you can set another value modally.
		> 0.0	The specified value is used.
		≤ 0.0	Not permitted
j	LREAL	-	Jerk[2]	
		-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Jerk". With setDyn() (Page 234), you can set another value modally.
		> 0.0	The specified value is used.
		= 0.0	No jerk limit
		< 0.0	Not permitted
trans	DINT	-	Motion transition to the next job[2]	
		-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
		0	Append motion
		1	Blend with the lower velocity of the two jobs.
		2	Blend with the higher velocity of the two jobs
blend	DINT	-	Blending mode[2]	

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

Parameter	Data type	Default value	Description	
blend	DINT	-	-	The modal value is used. By default, the "blend" parameter is preset to the value "2". With setBlend() (Page 250), you can set another value modally.
			0	Blending in path dynamics (direct blending)
			1	Reserved
			2	Geometric blending with polynomial path
blendDist	LREAL	-	Blending distance[2]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.
cs	DINT	-	Reference coordinate system[2]	
			-	The modal value is used. By default, the "cs" parameter is preset to the value "0". With setCs() (Page 279), you can set another value modally.
			0	World coordinate system (WCS)
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
			3	Object coordinate system 2 (OCS3)

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

Comparable Motion Control instructions

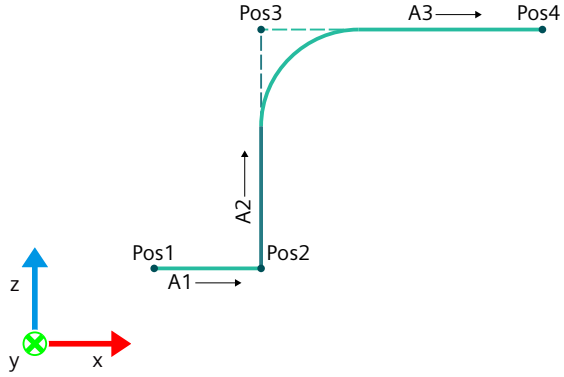
- "MC_MoveLinearRelative": Relative positioning of kinematics with linear path motion

Example: Relative positioning of kinematics with linear path motion

In the following example, a kinematics is to be moved from start position Pos1 with relative movement.

The kinematics should move from standstill Pos1 by Dist2 to Pos2, and stop at Pos2 (A1).

From Pos2, the kinematics is to move by Dist3 with the blending distance "blendingDist" via Pos3 (A2) by Dist4 to Pos4 (A3).



MCL

```
VAR
   Pos2, Pos3, Pos4 : TO_Struct_Ipr_Position;
   blendingDist : LREAL;
END_VAR
// A1 - Exact stop in position 2 with coordinate Pos2 relative to position 1
linRel( Pos2, trans := 0, cs := 0 );
// A2 - Blending high velocity in position 3 with coordinate Pos3 relative to position 2
linRel( Pos3, trans := 2, blend := 2, blendDist := blendingDist, cs := 0 );
// A3 - Exact stop in position 4 with coordinate Pos4 relative to position 3
linRel( Pos4, trans := 0, cs := 0 );
```

8.1.3 circAbs() Position kinematics absolutely with circular path motion (S7-1500T)

Description

With the MCL instruction "circAbs()", you move a kinematics with a circular motion to an absolute position.

The MCL instruction offers you the following options:

- Define circle path:
  - Define circle path via intermediate point and target position in space
  - Define circle path via center point and angle in a main plane
  - Define circle path via radius and target position in a main plane
- Define dynamics
- Define motion transition
- Define blending mode

The MCL instruction "circAbs()" ends when the absolute end position is reached by the kinematics or blending is started by the next job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
circAbs( <pos> [,auxPos := <val>] [,mode := <val>] [,arc := <val>] [,cr := <val>]
[,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] [,plane := <val>] [,cDir := <val>]
[,oDirA := <val>] [,trans := <val>] [,blend := <val>] [,blendDist := <val>] [,cs := <val>]
);
```

Parameters

The following table shows the parameters of the "circAbs()" instruction:

Parameter	Data type	Default value	Description
pos	TO_Struct_Ipr_Position	-	Absolute target position in the specified reference coordinate system If "mode" = 1, the target position is only relevant for the orientation
x	LREAL	0.0	x coordinate
y	LREAL	0.0	y coordinate
z	LREAL	0.0	z coordinate
A	LREAL	0.0	A coordinate
B	LREAL	0.0	B coordinate[1]
C	LREAL	0.0	C coordinate[1]
auxPos	TO_Struct_Ipr_Position	-	Absolute intermediate point of the circle path in the specified reference coordinate system[3] [4], if "mode" = 0
			Absolute center of the circle path[3] [4], if "mode" = 1
			Not relevant if "mode" = 2
x	LREAL	0.0	x coordinate
y	LREAL	0.0	y coordinate
z	LREAL	0.0	z coordinate
A	LREAL	0.0	A coordinate
B	LREAL	0.0	B coordinate[1]
C	LREAL	0.0	C coordinate[1]
mode	DINT	0	Interpolation type[3]
			0: 3D: Define circle path via intermediate point and target position
			1: 2D: Define circle path via center point and angle
			2: 2D: Define circle path via radius and target position
arc	LREAL	0.0	Angle of the circular motion[3] Only relevant for "mode" = 1
cr	LREAL	0.0	Radius of the circular motion[3] Only relevant for "mode" = 2
v	LREAL	-	Velocity[2]

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Parameter	Data type	Default value	Description	
v	LREAL	-	-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Velocity". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
a	LREAL	-	Acceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Acceleration". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
d	LREAL	-	Deceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Deceleration". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
j	LREAL	-	Jerk[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Jerk". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			= 0.0	No jerk limit
			< 0.0	Not permitted
plane	DINT	-	Main plane of the circle path[2] Only relevant for "mode" = 1 and "mode" = 2	
			-	The modal value is used. By default, the "plane" parameter is preset to the value "0". With setPlane() (Page 258), you can set another value modally.
			0	XZ plane
			1	YZ plane
			2	XY plane

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Parameter	Data type	Default value	Description	
cDir	DINT	-	Orientation of the circle path[2] Only relevant for "mode" = 1 and "mode" = 2	
			-	The modal value is used. By default, the "cDir" parameter is preset to the value "0". With setCircDir() (Page 264), you can set another value modally.
			0	Positive rotation direction (for "mode" = 1)
				Shorter positive circle segment (for "mode" = 2)
			1	Negative rotation direction (for "mode" = 1)
				Shorter negative circle segment (for "mode" = 2)
			2	Longer positive circle segment (for "mode" = 2)
			3	Longer negative circle segment (for "mode" = 2)
oDirA	DINT	-	Direction of movement of the Cartesian orientation A[2]. Only relevant for up to 4 interpolating kinematics axes and orientation axis with modulo functionality.	
			-	The modal value is used. By default, the "oDirA" parameter is preset to the value "3". With setOriDirA() (Page 260), you can set another value modally.
			1	Positive direction
			2	Negative direction
			3	Shortest path If the target orientation can be reached by two paths of the same length, the motion is in positive direction
trans	DINT	-	Motion transition to the next job[2]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
blend	DINT	-	Blending mode[2]	
			-	The modal value is used. By default, the "blend" parameter is preset to the value "2". With setBlend() (Page 250), you can set another value modally.
			0	Blending in path dynamics (direct blending)

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Parameter	Data type	Default value	Description	
blend	DINT	-	1	Reserved
			2	Geometric blending with polynomial path
blendDist	LREAL	-	Blending distance[2]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.
cs	DINT	-	Reference coordinate system[2]	
			-	The modal value is used. By default, the "cs" parameter is preset to the value "0". With setCs() (Page 279), you can set another value modally.
			0	World coordinate system (WCS)
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
			3	Object coordinate system 3 (OCS3)

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Comparable Motion Control instructions

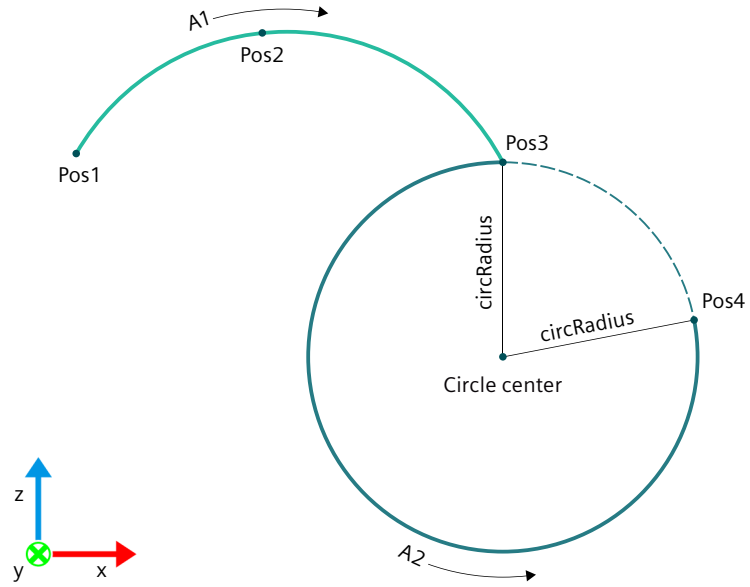
- "MC_MoveCircularAbsolute": Position kinematics with circular path motion

Example: Position kinematics with circular path motion

In the following example, a kinematics is to move to two positions.

The kinematics should move from standstill Pos1 (end point of the previous job) via Pos2 to Pos3, and stop at Pos3 (A1).

From Pos3, the kinematics should move with a defined circular path over a defined circular radius ("circRadius") and longer negative orientation of the circular path ("cDir" = 3) to Pos4 (A2).



MCL

```
VAR
   Pos1, Pos2, Pos3, Pos4 : TO_Struct_Ipr_Position;
   circRadius : LREAL;
END_VAR
// A1 - circular movement through Pos2 with defined dynamics, exact stop in Pos3
circAbs( Pos3, auxPos := Pos2, mode := 0, v := 15.0, a := 2000.0, d := 3000.0, j :=
10000.0, trans := 0, cs := 0 );
// A2 - circular movement to Pos4 with defined radius and longer negative circle segment
// the modal values of dynamics will be used
circAbs( Pos4, mode := 2, cr := circRadius, plane := 0, cDir := 3, trans := 0, cs := 0 );
```

8.1.4 circRel(): Relative positioning of kinematics with circular path motion (S7-1500T)

Description

With the MCL instruction "circRel()", you move a kinematics with a circular motion relative to the position which was present at the start of the job processing.

The MCL instruction offers you the following options:

- Define circle path:
  - Define circle path via intermediate point and target position in space
  - Define circle path via center point and angle in a main plane
  - Define circle path via radius and target position in a main plane
- Define dynamics
- Define motion transition
- Define blending mode

The MCL instruction "circRel()" ends when the relative end position is reached by the kinematics or blending is started by the next job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
circRel( <pos> [,auxPos := <val>] [,mode := <val>] [,arc := <val>] [,cr := <val>]
[,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] [,plane := <val>]
[,cDir := <val>] [,trans := <val>] [,blend := <val>] [,blendDist := <val>] [,cs := <val>]
);
```

Parameters

The following table shows the parameters of the instruction "circRel()":

Parameter	Data type	Default value	Description	
pos	TO_Struct_Ipr_Position	-	Target position relative to the start position in the specified reference coordinate system If "mode" = 1, the target position is only relevant for the orientation	
x	LREAL	0.0	x coordinate	
y	LREAL	0.0	y coordinate	
z	LREAL	0.0	z coordinate	
A	LREAL	0.0	A coordinate	
B	LREAL	0.0	B coordinate[1]	
C	LREAL	0.0	C coordinate[1]	
auxPos	TO_Struct_Ipr_Position	-	Intermediate point on the circle path relative to the start position[3] [4], if "mode" = 0	
			Center of the circle path in reference to the start position[3] [4] , if "mode" = 1	
			Not relevant if "mode" = 2	
x	LREAL	0.0	x coordinate	
y	LREAL	0.0	y coordinate	
z	LREAL	0.0	z coordinate	
A	LREAL	0.0	A coordinate	
B	LREAL	0.0	B coordinate[1]	
C	LREAL	0.0	C coordinate[1]	
mode	DINT	0	Interpolation type[3]	
			0	3D: Define circle path via intermediate point and target position
			1	2D: Define circle path via center point and angle
			2	2D: Define circle path via radius and target position

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Parameter	Data type	Default value	Description	
arc	LREAL	0.0	Angle of the circular motion[3] Only relevant for "mode" = 1	
cr	LREAL	0.0	Radius of the circular motion[3] Only relevant for "mode" = 2	
v	LREAL	-	Velocity[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Velocity". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
a	LREAL	-	Acceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Acceleration". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
d	LREAL	-	Deceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Deceleration". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
j	LREAL	-	Jerk[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.Path.Jerk". With setDyn() (Page 234), you can set another value modally.
			> 0.0	The specified value is used.
			= 0.0	No jerk limit
			< 0.0	Not permitted
plane	DINT	-	Main plane of the circle path[2] Only relevant for "mode" = 1 and "mode" = 2	

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Parameter	Data type	Default value	Description	
plane	DINT	-	-	The modal value is used. By default, the "plane" parameter is preset to the value "0". With setPlane() (Page 258), you can set another value modally.
			0	XZ plane
			1	YZ plane
			2	XY plane
cDir	DINT	-	Orientation of the circle path[2] Only relevant for "mode" = 1 and "mode" = 2	
			-	The modal value is used. By default, the "cDir" parameter is preset to the value "0". With setCircDir() (Page 264), you can set another value modally.
			0	Positive rotation direction (for "mode" = 1)
				Shorter positive circle segment (for "mode" = 2)
			1	Negative rotation direction (for "mode" = 1)
				Shorter negative circle segment (for "mode" = 2)
			2	Longer positive circle segment (for "mode" = 2)
			3	Longer negative circle segment (for "mode" = 2)
trans	DINT	-	Motion transition to the next job[2]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
blend	DINT	-	Blending mode[2]	
			-	The modal value is used. By default, the "blend" parameter is preset to the value "2". With setBlend() (Page 250), you can set another value modally.
			0	Blending in path dynamics (direct blending)
			1	Reserved
			2	Geometric blending with polynomial path
blendDist	LREAL	-	Blending distance[2]	

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Parameter	Data type	Default value	Description	
blendDist	LREAL	-	-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.
cs	DINT	-	Reference coordinate system[2]	
			-	The modal value is used. By default, the "cs" parameter is preset to the value "0". With setCs() (Page 279), you can set another value modally.
			0	World coordinate system (WCS)
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
			3	Object coordinate system 3 (OCS3)

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

4) Only relevant for Cartesian coordinates, orientation is ignored.

Comparable Motion Control instructions

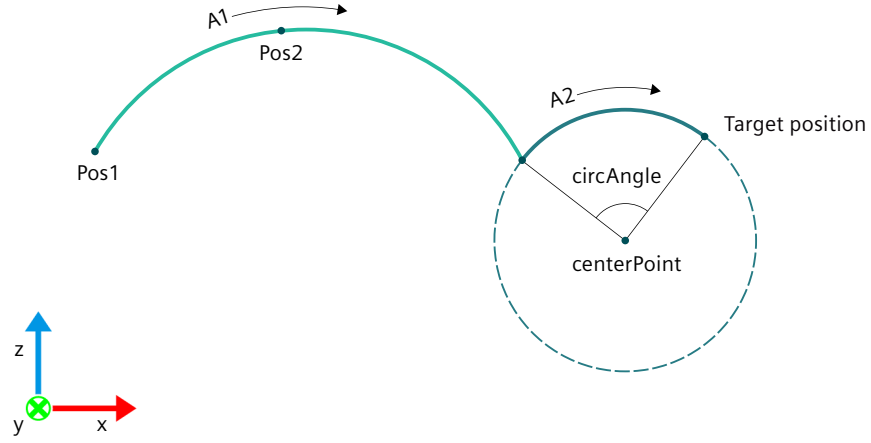
- "MC_MoveCircularRelative": Relative positioning of kinematics with circular path motion

Example: Relative positioning of kinematics with circular path motion

In the following example, a kinematics is to move to two positions.

The kinematics should move from standstill Pos1 (end point of the previous job) between Pos2 to Pos3, and stop at Pos3 (A1). The coordinates for Pos2 and Pos3 are relative to the start position, Pos1.

From Pos3, the kinematics should move to the target position, which is defined by the center of the circle path ("centerPoint" relative to Pos3) and angle of the circular motion ("circAngle"), with a shorter positive circular segment orientation of the circle path ("cDir" = 0) (A2).



MCL

```
VAR
   Pos1, Pos2, Pos3, centerPoint : TO_Struct_Ipr_Position;
   circAngle : LREAL;
END_VAR
// A1 - circular movement through Pos2 with defined dynamics, exact stop in Pos3
circRel( Pos3, auxPos := Pos2, mode := 0, trans := 0, cs := 0 );

//A2 - circular movement to the target position, defined by circle center point and angle

// the modal values of dynamics will be used
circRel( (A := 0.0, B := 0.0, C := 0.0), auxPos := centerPoint, mode := 1, arc :=
circAngle,
      cDir := 0, trans := 0, cs := 0 );
```

8.1.5 ptpAbs() Moving kinematics to absolute Cartesian target coordinates with sPTP motion (S7-1500T)

Description

With the MCL instruction "ptpAbs()", you can move kinematics to absolute positions with a synchronous "point-to-point" (sPTP) motion. All kinematics axes are moved at the same time. The axes start the movement at the same time and reach the target position at the same time.

The movement path of the tool center point (TCP) results from the dynamic values of the axes. The kinematics axis with the longest travel time determines the travel time of the sPTP motion and therefore the travel time of all other kinematics axes. The position of the TCP results from the positions of the kinematics axes.

The MCL instruction offers you the following options:

- Define absolute target position (depending on the kinematics, this means a Cartesian specification of the target position (x, y, z) and an orientation in space (A, B, C))
- Define dynamics
- Define motion transition
- Define blending distance

The MCL instruction "ptpAbs()" ends when the absolute end position is reached by the kinematics, or blending is started by the job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
ptpAbs( <pos> [,posMode := <val>] [,lc := <val>] [,cs := <val>] [,v := <val>]
[,a := <val>] [,d := <val>] [,j := <val>] [,trans := <val>] [,blendDist := <val>]
[,oDirA := <val>] [,tj1 := <val>] [,tj2 := <val>] [,tj3 := <val>] [,tj4 := <val>]
[,tj5 := <val>] [,tj6 := <val>] );
```

Parameters

The following table shows the parameters of the "ptpAbs()" instruction:

Parameter	Data type	Default value	Description	
pos	TO_Struct_Ipr_Position	-	Absolute Cartesian target position in the specified reference coordinate system ("posMode" = 1). Absolute Cartesian target position and relative Cartesian orientation ("posMode" = 2).	
x	LREAL	0.0	x coordinate	
y	LREAL	0.0	y coordinate	
z	LREAL	0.0	z coordinate	
A	LREAL	0.0	A coordinate	
B	LREAL	0.0	B coordinate[1]	
C	LREAL	0.0	C coordinate[1]	
posMode	DINT	1	Type of orientation A for absolute Cartesian target position ("pos")[3]	
			1	Absolute Cartesian target position and absolute Cartesian orientation A
			2	Absolute Cartesian target position and relative Cartesian orientation A
			Only relevant for kinematics types with orientation A. With more than 4 interpolating kinematics axes: Not relevant	
lc	DWORD	-	Target joint position space[2] The joint position space is specified depending on the respective kinematics type.	
			-	The modal value is used. By default, the "lc" parameter is preset to the value "16#FFFF_FFFF". With setLc() (Page 268), you can set another value modally.
			16#0000_0000 ≤ lc < 16#FFFF_FFFF	The specified value is interpreted as a bit field. The meaning of the bits depends on the type of kinematics. 6 axes ($A1, $A2, $A3, $A4, $A5, $A6) correspond to the first 6 bits (bit 0 ... bit 5). All other bits are ignored.
			16#FFFF_FFFF	Keep current joint position space
cs	DINT	-	Reference coordinate system[2]	

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

Parameter	Data type	Default value	Description	
cs	DINT	-	-	The modal value is used. By default, the "cs" parameter is preset to the value "0". With setCs() (Page 279), you can set another value modally.
			0	World coordinate system (WCS)
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
			3	Object coordinate system 2 (OCS3)
v	LREAL	-	Percentage factor for the velocity of the axis movement in relation to the respective maximum velocity of the axes ("<TO_Axis>.DynamicLimits.MaxVelocity")[2].	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor". With setPtpDyn() (Page 240), you can set another value modally.
			1.0 ≤ v ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			< 1.0	Not permitted
a	LREAL	-	Percentage factor for acceleration of the axis movements in relation to the respective maximum acceleration of the axes ("<TO_Axis>.DynamicLimits.MaxAcceleration")[2].	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < a ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
d	LREAL	-	Percentage factor for the deceleration of the axis movement in relation to the respective maximum deceleration of the axes ("<TO_Axis>.DynamicLimits.MaxDeceleration")[2].	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < d ≤ 100.0	The specified value is used.
			> 100.0	Not permitted

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

[3] Optional parameter with default value

Parameter	Data type	Default value	Description	
d	LREAL	-	≤ 0.0	Not permitted
j	LREAL	-	Percentage factor for jerk of axis movements in relation to the respective maximum jerk of the axes ("<TO_Axis>.DynamicLimits.MaxJerk")[2].	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor". With setPtpDyn() (Page 240), you can set another value modally.
			10.0 ≤ j ≤ 90.0	The specified value is used.
			= 0.0	No jerk limit
			< 10.0 (except 0.0)	Not permitted
			> 90.0	Not permitted
trans	DINT	-	Motion transition to the next job[2]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
			If "trans" = 1 or "trans" = 2: Always polynomial blending between sPTP and path motions.	
blendDist	LREAL	-	Blending distance[2]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.
oDirA	DINT	-	Direction of movement of the Cartesian orientation A[2]. Only relevant for: - Up to 4 interpolating kinematics axes - "posMode" = 1 - Kinematics types with orientation A and activated modulo functionality	
			-	The modal value is used. By default, the "oDirA" parameter is preset to the value "3". With setOriDirA() (Page 260), you can set another value modally.

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

Parameter	Data type	Default value	Description	
oDirA	DINT	-	1	Positive direction
			2	Negative direction
			3	Shortest distance If the target orientation can be reached by two paths of the same length, the motion is in a positive direction.
tj1,...,tj6	DINT	-	Target joint position range of the J1, ..., J6[2] Only relevant for kinematics whose joint coordinate system supports target joint position ranges.	
			-	The modal value is used. By default, the "tj1",..., "tj6" parameter is preset to the value "0". With setTurnJoint() (Page 262), you can set another value modally.
			m (m < 0)	-180° + m * 360° ≤ Position < 180° + m * 360°
			...	...
			-2	-900° ≤ Position < -540°
			-1	-540° ≤ Position < -180°
			0	Shortest distance
			1	-180° ≤ Position < 180°
			2	180° ≤ Position < 540°
			3	540° ≤ Position < 900°
			...	...
			n (n > 0)	-180° + (n - 1) * 360° ≤ Position < 180° + (n - 1) * 360°

1) Only relevant with more than 4 interpolating kinematics axes.

2) Modal parameter

3) Optional parameter with default value

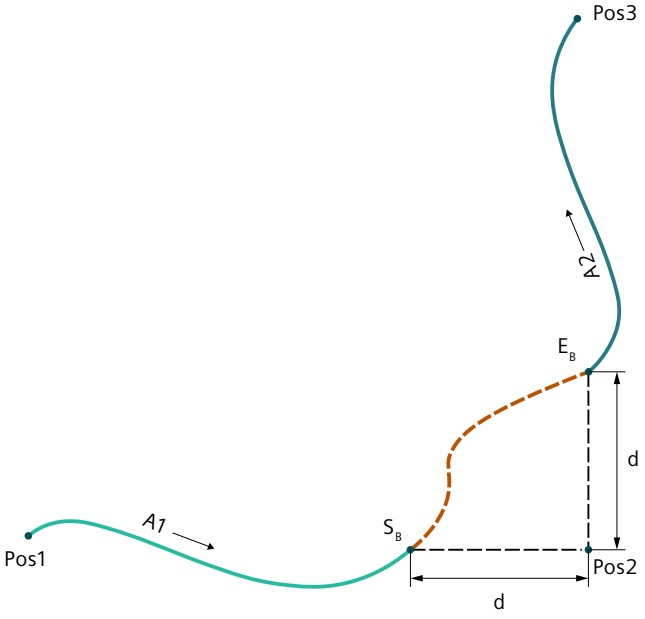
Comparable Motion Control instructions

- "MC_MoveDirectAbsolute": Absolute movement of kinematics with synchronous "point-to-point" motion

Example: Positioning kinematics to absolute Cartesian target coordinates with sPTP motion

In the following example, a kinematics is to move to two absolute positions.

The kinematics is to move from a standstill from Pos1 (end point of the previous job) to Pos3 (A2) via Pos2 (A1) with blending with high velocity ("trans" = 2) using the blending distance ("blendDist" = d).



S_B Start of blending segment

E_B End of blending segment

MCL

```
VAR
   Pos2, Pos3 : TO_Struct_Ipr_Position;
   d : LREAL;
END_VAR

// A1 - sPTP movement with blending with high velocity through Pos2
// max. sPTP dynamics without jerk limitation and coord. system WCS
ptpAbs( Pos2, cs := 0, v := 100.0, a := 100.0, d := 100.0,
        j := 0.0, trans := 2, blendDist := d );

// A2 - sPTP movement to Pos3, without blending and modal dynamics
ptpAbs( Pos3, cs := 0, trans := 0 );
```

8.1.6 ptpRel() Move kinematics to relative Cartesian target coordinates with sPTP motion (S7-1500T)

Description

With the MCL instruction "ptpRel()", you can move a kinematics in a synchronous "point-to-point" (sPTP) motion to relative positions. All kinematics axes are moved at the same time. The axes start the movement at the same time and reach the target position at the same time.

The movement path of the tool center point (TCP) results from the dynamic values of the axes. The kinematics axis with the longest travel time determines the travel time of the sPTP motion and therefore the travel time of all other kinematics axes. The position of the TCP results from the positions of the kinematics axes.

The MCL instruction offers you the following options:

- Define relative target position (depending on the kinematics, this means a Cartesian specification of the target position (x, y, z) and an orientation in space (A, B, C))
- Define dynamics
- Define motion transition
- Define blending distance

The MCL instruction "ptpRel()" ends when the relative end position is reached by the kinematics, or blending is started by the job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
ptpRel( <pos> [,lc := <val>] [,cs := <val>] [,v := <val>] [,a := <val>] [,d := <val>]
[,j := <val>] [,trans := <val>] [,blendDist := <val>] [,tj1 := <val>] [,tj2 := <val>]
[,tj3 := <val>] [,tj4 := <val>] [,tj5 := <val>] [,tj6 := <val>] );
```

Parameters

The following table shows the parameters of the "ptpRel()" instruction:

Parameter	Data type	Default value	Description	
pos	TO_Struct_Ipr_Position	-	Relative Cartesian target position in the specified reference coordinate system	
x	LREAL	0.0	x coordinate	
y	LREAL	0.0	y coordinate	
z	LREAL	0.0	z coordinate	
A	LREAL	0.0	A coordinate	
B	LREAL	0.0	B coordinate[1]	
C	LREAL	0.0	C coordinate[1]	
lc	DWORD	-	Target joint position space[2] The joint position space is determined depending on the type of kinematics.	
			-	The modal value is used. By default, the "lc" parameter is preset to the value "16#FFFF_FFFF". With setLc() (Page 268), you can set another value modally.
			16#0000_0000 ≤ lc < 16#FFFF_FFFF	The specified value is interpreted as a bit field. The meaning of the bits depends on the type of kinematics. 6 axes ($A1, $A2, $A3, $A4, $A5, $A6) correspond to the first 6 bits (bit 0 ... bit 5). All other bits are ignored.
			16#FFFF_FFFF	Keep current joint position space
cs	DINT	-	Reference coordinate system[2]	
			-	The modal value is used. By default, the "cs" parameter is preset to the value "0". With setCs() (Page 279), you can set another value modally.
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
			3	Object coordinate system 2 (OCS3)
v	LREAL	-	Percentage factor for the velocity of the axis movements in relation to the respective maximum velocity of the axes ("<TO_Axis>.DynamicLimits.MaxVelocity") [2]	

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

Parameter	Data type	Default value	Description	
v	LREAL	-	-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor". With setPtpDyn() (Page 240), you can set another value modally.
			1.0 ≤ v ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			< 1.0	Not permitted
a	LREAL	-	Percentage factor for acceleration of the axis movements in relation to the respective maximum acceleration of the axes ("<TO_Axis>.DynamicLimits.MaxAcceleration")[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < a ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
d	LREAL	-	Percentage factor for deceleration of the axis movements in relation to the respective maximum deceleration of the axes ("<TO_Axis>.DynamicLimits.MaxDeceleration")[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < d ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
j	LREAL	-	Percentage factor for the jerk of the axis movements in relation to the respective maximum jerk of the axes ("<TO_Axis>.DynamicLimits.MaxJerk")[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor". With setPtpDyn() (Page 240), you can set another value modally.
			10.0 ≤ j ≤ 90.0	The specified value is used.
			= 0.0	No jerk limit
			< 10.0 (except 0.0)	Not permitted

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

Parameter	Data type	Default value	Description	
j	LREAL	-	> 90.0	Not permitted
trans	DINT	-	Motion transition to the next job[2]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
			If "trans" = 1 or "trans" = 2: Always polynomial blending between sPTP and path motions.	
blendDist	LREAL	-	Blending distance[2]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			< 0.0	The maximum possible blending distance is used.
tj1,...,tj6	DINT	-	Target joint position range of joints J1,...,J6[2] Only relevant for kinematics whose joint coordinate system supports target joint position ranges.	
			-	The modal value is used. By default, the "tj1",..., "tj6" parameter is preset to the value "0". With setTurnJoint() (Page 262), you can set another value modally.
			m (m < 0)	-180° + m * 360° ≤ Position < 180° + m * 360°
			...	...
			-2	-900° ≤ Position < -540°
			-1	-540° ≤ Position < -180°
			0	Shortest distance
			1	-180° ≤ Position < 180°
			2	180° ≤ Position < 540°
			3	540° ≤ Position < 900°
			...	...
			n (n > 0)	-180° + (n - 1) * 360° ≤ Position < 180° + (n - 1) * 360°

[1] Only relevant with more than 4 interpolating kinematics axes.

[2] Modal parameter

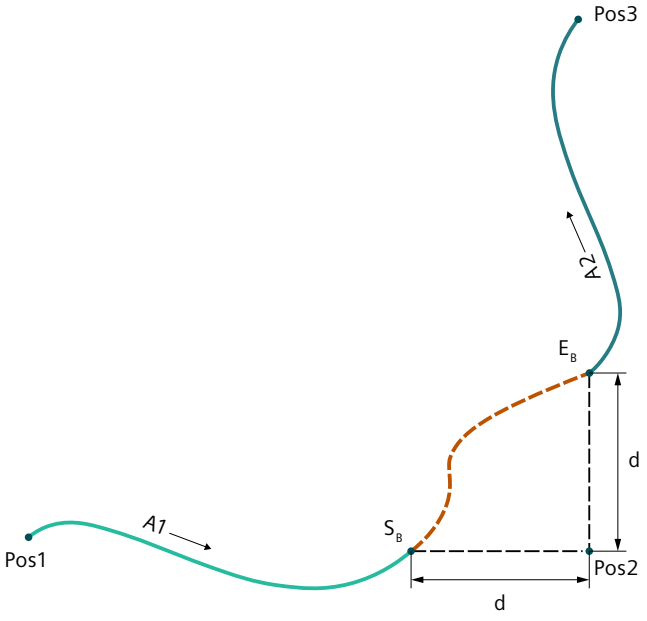
Comparable Motion Control instructions

- "MC_MoveDirectRelative": Relative movement of kinematics with synchronous "point-to-point" motion

Example: Positioning kinematics to relative Cartesian target coordinates with sPTP motion

In the following example, a kinematics is to move to two relative positions.

The kinematics is to move from a standstill with relative moment from Pos1 (end point of the previous job) to Pos3 via Pos2 (A1) with blending with low velocity ("trans" = 1) using the blending distance ("blendDist" = d) (A1).



S_B Start of blending segment

E_B End of blending segment

MCL

```
VAR
   Pos2, Pos3 : TO_Struct_Ipr_Position;
   d : LREAL;
END_VAR

// A1 - sPTP movement with blending with low velocity through Pos2 with coordinate Pos2
// relative to Pos1
ptpRel( Pos2, cs := 0, v := 100.0, a := 100.0, d := 100.0, j := 0.0,
        trans := 1, blendDist := d );

// A2 - sPTP movement to Pos3 with coordinate Pos3 relative to Pos2
// without blending and modal dynamics
ptpRel( Pos3, cs := 0, trans := 0 );
```

8.1.7 ptpAxAbs() Move kinematics to absolute axis positions with sPTP motion (S7-1500T)

Description

With the MCL instruction "ptpAxAbs()", you can move kinematics in a synchronous "point-to-point" (sPTP) motion to absolute axis positions in the machine coordinate system. All kinematics axes are moved at the same time. The axes start the movement at the same time and reach the axis target position at the same time.

The movement path of the tool center point (TCP) results from the dynamic values of the axes. The kinematics axis with the longest travel time determines the travel time of the sPTP motion and therefore the travel time of all other kinematics axes. The position of the TCP results from the positions of the kinematics axes.

The MCL instruction offers you the following options:

- Define absolute axis target position in the machine coordinate system
- Define dynamics
- Define motion transition
- Define blending distance

The MCL instruction "ptpAxAbs()" ends when the absolute end position is reached by the kinematics, or blending is started by the job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
ptpAxAbs( <axPos> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] [,trans := <val>]
[,blendDist := <val>] );
```

Parameters

The following table shows the parameters of the "ptpAxAbs()" instruction:

Parameter	Data type	Default value	Description	
axPos	TO_Struct_Ipr_AxPosition	-	Absolute axis target position in the machine coordinate system (MCS)	
a1	LREAL	0.0	Absolute target position of the kinematics axis A1 in the MCS (axis coordinates)	
a2	LREAL	0.0	Absolute target position of the kinematics axis A2 in the MCS (axis coordinates)	
a3	LREAL	0.0	Absolute target position of the kinematics axis A3 in the MCS (axis coordinates)	
a4	LREAL	0.0	Absolute target position of the kinematics axis A4 in the MCS (axis coordinates)	
a5	LREAL	0.0	Absolute target position of the kinematics axis A5 in the MCS (axis coordinates)	
a6	LREAL	0.0	Absolute target position of the kinematics axis A6 in the MCS (axis coordinates)	
v	LREAL	-	Percentage factor for the velocity of the axis movements in relation to the respective maximum velocity of the axes (<TO_Axis>.DynamicLimits.MaxVelocity) [1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor". With setPtpDyn() (Page 240), you can set another value modally.
			1.0 ≤ v ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			< 1.0	Not permitted
a	LREAL	-	Percentage factor for acceleration of the axis movements in relation to the respective maximum acceleration of the axes (<TO_Axis>.DynamicLimits.MaxAcceleration)[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < a ≤ 100.0	The specified value is used.
			> 100.0	Not permitted

[1] Modal parameter

Parameter	Data type	Default value	Description	
a	LREAL	-	≤ 0.0	Not permitted
d	LREAL	-	Percentage factor for the deceleration of the axis movements in relation to the respective maximum deceleration of the axes (<TO_Axis>.DynamicLimits.MaxDeceleration)[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < d ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
j	LREAL	-	Percentage factor for the jerk of the axis movements in relation to the respective maximum jerk of the axes (<TO_Axis>.DynamicLimits.MaxJerk)[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor". With setPtpDyn() (Page 240), you can set another value modally.
			10.0 ≤ j ≤ 90.0	The specified value is used.
			= 0.0	No jerk limit
			< 10.0 (except 0.0)	Not permitted
			> 90.0	Not permitted
trans	DINT	-	Motion transition to the next job[1]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
			If "trans" = 1 or "trans" = 2: Always polynomial blending between sPTP and path motions.	
blendDist	LREAL	-	Blending distance[1]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.

[1] Modal parameter

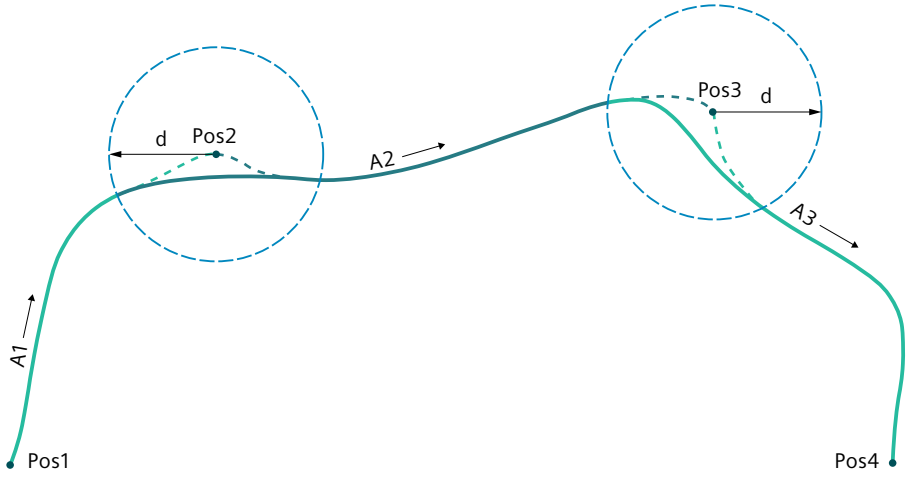
Comparable Motion Control instructions

- "MC_MoveDirectAbsolute": Absolute movement of kinematics with synchronous "point-to-point" motion (if "CoordSystem" = 100)

Example: Positioning kinematics with sPTP motion to absolute axis target position in the machine coordinate system

In the following example, a kinematics is to move to three absolute axis target positions in the machine coordinate system.

The kinematics should start from standstill from position 1 (end point of the previous command) via position 2 with blending at high velocity ("trans" = 2) using the blending distance ("blendDist" = d) (A1), move via position 3 with blending at low velocity ("trans" = 1) using the blending distance ("blendDist" = d) (A2), and append motion ("trans" = 0) to position 4 (A3).

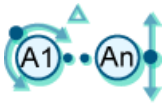


MCL

```
VAR
   Pos2, Pos3, Pos4 : TO_Struct_Ipr_AxPosition;
   d : LREAL;
END_VAR
// A1 - sPTP movement with blending with high velocity
// through Pos2 (abs. target Pos in MCS)
// with max. sPTP dynamics without jerk limitation
ptpAxAbs( Pos2, v := 100.0, a := 100.0, d := 100.0, j := 0.0, trans := 2, blendDist := d );
// A2 - sPTP movement with blending with low velocity
// through Pos3 (abs. target Pos in MCS)
// with defined sPTP dynamics without jerk limitation
ptpAxAbs( Pos3, v := 75.0, a := 80.0, d := 90.0, j := 0.0, trans := 1, blendDist := d );
// A3 - sPTP movement to Pos4 in machine coordinate system, without blending
ptpAxAbs( Pos4, trans := 0 );
```

8.1.8 ptpAxRel() Move kinematics to relative axis positions with sPTP motion (S7-1500T)

Description



With the MCL instruction "ptpAxRel()", you can move kinematics in a synchronous "point-to-point" (sPTP) motion to a relative axis target position in the machine coordinate system. All kinematics axes are moved at the same time. The axes start the movement at the same time and reach the axis target position at the same time.

The movement path of the tool center point (TCP) results from the dynamic values of the axes. The kinematics axis with the longest travel time determines the travel time of the sPTP motion and therefore the travel time of all other kinematics axes. The position of the TCP results from the positions of the kinematics axes.

The MCL instruction offers you the following options:

- Define relative axis positions in the machine coordinate system
- Define dynamics
- Define motion transition
- Define blending distance

The MCL instruction "ptpAxRel()" ends when the relative end position is reached by the kinematics, or blending is started by the job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
ptpAxRel( <axPos> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] [,tr:= <val>]
[,blendDist := <val>] );
```

Parameters

The following table shows the parameters of the "ptpAxRel()" instruction:

Parameter	Data type	Default value	Description	
axPos	TO_Struct_Ipr_AxPosition	-	Relative axis target position in the machine coordinate system (MCS)	
a1	LREAL	0.0	Relative target position of the kinematics axis A1 in the MCS (axis coordinates)	
a2	LREAL	0.0	Relative target position of the kinematics axis A2 in the MCS (axis coordinates)	
a3	LREAL	0.0	Relative target position of the kinematics axis A3 in the MCS (axis coordinates)	
a4	LREAL	0.0	Relative target position of the kinematics axis A4 in the MCS (axis coordinates)	
a5	LREAL	0.0	Relative target position of the kinematics axis A5 in the MCS (axis coordinates)	
a6	LREAL	0.0	Relative target position of the kinematics axis A6 in the MCS (axis coordinates)	
v	LREAL	-	Percentage factor for the velocity of the axis movements in relation to the respective maximum velocity of the axes ("<TO_Axis>.DynamicLimits.MaxVelocity") [1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor". With setPtpDyn() (Page 240), you can set another value modally.
			1.0 ≤ v ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			< 1.0	Not permitted
a	LREAL	-	Percentage factor for the acceleration of the axis movements in relation to the respective maximum acceleration of the axes ("<TO_Axis>.DynamicLimits.MaxAcceleration")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < a ≤ 100.0	The specified value is used.
			> 100.0	Not permitted

[1] Modal parameter

Parameter	Data type	Default value	Description	
a	LREAL	-	≤ 0.0	Not permitted
d	LREAL	-	Percentage factor for the deceleration of the axis movements in relation to the respective maximum deceleration of the axes ("<TO_Axis>.DynamicLimits.MaxDeceleration")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < d ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
j	LREAL	-	Percentage factor for the jerk of the axis movements in relation to the respective maximum jerk of the axes ("<TO_Axis>.DynamicLimits.MaxJerk")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor". With setPtpDyn() (Page 240), you can set another value modally.
			10.0 ≤ j ≤ 90.0	The specified value is used.
			= 0.0	No jerk limit
			< 10.0 (except 0.0)	Not permitted
			> 90.0	Not permitted
trans	DINT	-	Motion transition to the next job[1]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
			If "trans" = 1 or "trans" = 2: Always polynomial blending between sPTP and path motions.	
blendDist	LREAL	-	Blending distance[1]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.

[1] Modal parameter

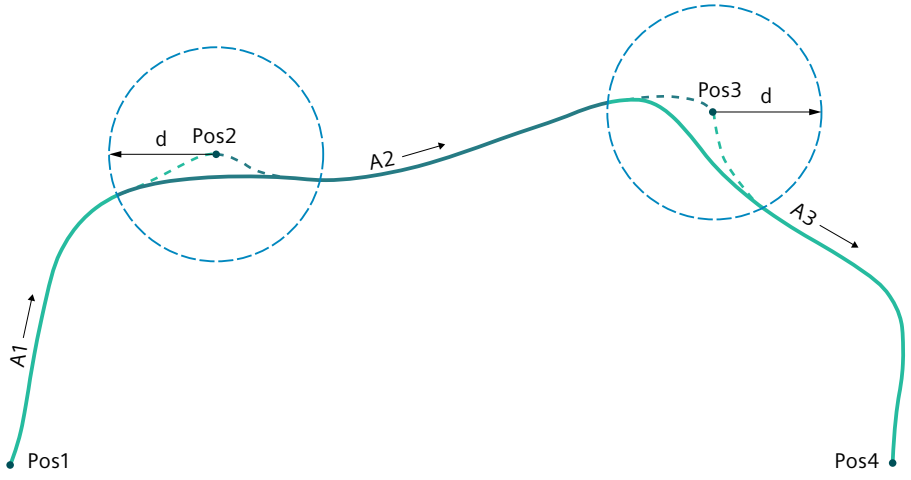
Comparable Motion Control instructions

- "MC_MoveDirectRelative": Relative movement of kinematics with synchronous "point-to-point" motion (if "CoordSystem" = 100)

Example: Positioning kinematics with sPTP motion to relative axis target position in the machine coordinate system

In the following example, a kinematics is to move to three relative axis positions in the machine coordinate system.

The kinematics should start from standstill from position 1 (end point of the previous command) via position 2 with blending at high velocity ("trans" = 2) using the blending distance ("blendDist" = d) (A1), move via position 3 with blending at low velocity ("trans" = 1) using the blending distance ("blendDist" = d) (A2), and append motion ("trans" = 0) to position 4 (A3).



MCL

```
VAR
   Pos2, Pos3, Pos4 : TO_Struct_Ipr_AxPosition;
   d : LREAL;
END_VAR
// A1 - sPTP movement with blending with high velocity
// through Pos2 with coordinate Pos2
// relative to Pos1 in MCS with max. sPTP dynamics without jerk limitation
ptpAxRel( Pos2, v := 100.0, a := 100.0, d := 100.0, j := 0.0, trans := 2, blendDist := d );
// A2 - sPTP movement with blending with low velocity through Pos3 with coordinate Pos3
// relative to Pos2 in MCS with defined sPTP dynamics without jerk limitation
ptpAxRel( Pos3, v := 75.0, a := 80.0, d := 90.0, j := 0.0, trans := 1, blendDist := d );
// A3 - sPTP movement to Pos4 with coordinate Pos4
// relative to Pos3 in MCS, without blending
ptpAxRel( Pos4, trans := 0 );
```

8.1.9 ptpJtAbs() Move kinematics to absolute joint positions with sPTP motion (S7-1500T)

Description

With the MCL instruction "ptpJtAbs()", you can move kinematics in a synchronous "point-to-point" (sPTP) motion to absolute joint target positions in the joint coordinate system (JCS). All kinematics axes are moved at the same time. The axes start the movement at the same time and reach the joint target position at the same time.

The movement path of the tool center point (TCP) results from the dynamic values of the axes. The kinematics axis with the longest travel time determines the travel time of the sPTP motion and therefore the travel time of all other kinematics axes. The position of the TCP results from the positions of the kinematics axes.

The MCL instruction offers you the following options:

- Define absolute joint target positions in the joint coordinate system (JCS)
- Define dynamics
- Define motion transition
- Define blending distance

The MCL instruction "ptpJtAbs()" ends when the absolute end position is reached by the kinematics, or blending is started by the job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
ptpJtAbs( <jtPos> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] [,trans := <val>]
[,blendDist := <val>] );
```

Parameters

The following table shows the parameters of the "ptpJtAbs()" instruction:

Parameter	Data type	Default value	Description	
jtPos	TO_Struct_Ipr_JtPosition	-	Absolute joint target positions in the joint coordinate system (JCS)	
j1	LREAL	0.0	Absolute target position of joint J1 in the JCS	
j2	LREAL	0.0	Absolute target position of joint J2 in the JCS	
j3	LREAL	0.0	Absolute target position of joint J3 in the JCS	
j4	LREAL	0.0	Absolute target position of joint J4 in the JCS	
j5	LREAL	0.0	Absolute target position of joint J5 in the JCS	
j6	LREAL	0.0	Absolute target position of joint J6 in the JCS	
v	LREAL	-	Percentage factor for the velocity of the axis movements in relation to the respective maximum velocity of the axes ("<TO_Axis>.DynamicLimits.MaxVelocity") [1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor". With setPtpDyn() (Page 240), you can set another value modally.
			1.0 ≤ v ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			< 1.0	Not permitted
a	LREAL	-	Percentage factor for the acceleration of the axis movements in relation to the respective maximum acceleration of the axes ("<TO_Axis>.DynamicLimits.MaxAcceleration")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < a ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
d	LREAL	-	Percentage factor for the deceleration of the axis movements in relation to the respective maximum deceleration of the axes ("<TO_Axis>.DynamicLimits.MaxDeceleration")[1]	

[1] Modal parameter

Parameter	Data type	Default value	Description	
d	LREAL	-	-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < d ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
j	LREAL	-	Percentage factor for the jerk of the axis movements in relation to the respective maximum jerk of the axes ("<TO_Axis>.DynamicLimits.MaxJerk")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor". With setPtpDyn() (Page 240), you can set another value modally.
			10.0 ≤ j ≤ 90.0	The specified value is used.
			= 0.0	No jerk limit
			< 10.0 (except 0.0)	Not permitted
			> 90.0	Not permitted
trans	DINT	-	Motion transition to the next job[1]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
			If "trans" = 1 or "trans" = 2: Always polynomial blending between sPTP and path motions.	
blendDist	LREAL	-	Blending distance[1]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.

[1] Modal parameter

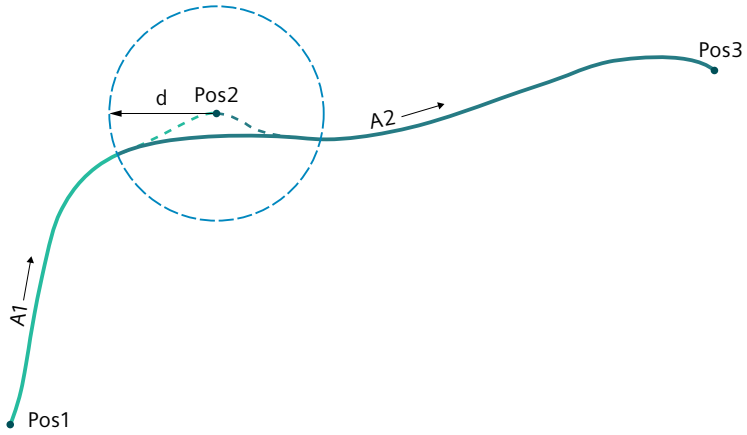
Comparable Motion Control instructions

- "MC_MoveDirectAbsolute": Relative movement of kinematics with synchronous "point-to-point" motion (if "CoordSystem" = 101)

Example: Positioning kinematics with sPTP motion at absolute joint positions in the joint coordinate system

In the following example, a kinematics is to move to two absolute joint target positions in the joint coordinate system.

The kinematics should move from standstill Position 1 with blending distance ("blendDist" = d) and at high velocity ("trans" = 2) via Position 2 (A1) to Position 3 (A2) ("trans" = 0).



MCL

```
VAR
   Pos2, Pos3 : TO_Struct_Ipr_JtPosition;
   d : LREAL;
END_VAR
// A1 - sPTP movement with blending
// with high velocity in Pos2 (absolute target position in JCS)
// with max. sPTP dynamics without jerk limitation
ptpJtAbs( Pos2, v := 100.0, a := 100.0, d := 100.0, j := 0.0, trans := 2, blendDist := d );
// A2 - sPTP movement without blending to Pos3 (abs. target position in JCS)
// Exact stop in Pos3
ptpJtAbs( Pos3, trans := 0 );
…
```

8.1.10 ptpJtRel() Move kinematics to relative joint positions with sPTP motion (S7-1500T)

Description

With the MCL instruction "ptpJtRel()", you can move kinematics in a synchronous "point-to-point" (sPTP) motion to relative joint target positions in the joint coordinate system (JCS). All kinematics axes are moved at the same time. The axes start the movement at the same time and reach the joint target position at the same time.

The movement path of the tool center point (TCP) results from the dynamic values of the axes. The kinematics axis with the longest travel time determines the travel time of the sPTP motion and therefore the travel time of all other kinematics axes. The position of the TCP results from the positions of the kinematics axes.

The MCL instruction offers you the following options:

- Define relative joint target positions in the joint coordinate system (JCS)
- Define dynamics
- Define motion transition
- Define blending distance

The MCL instruction "ptpJtRel()" ends when the relative end position is reached by the kinematics, or blending is started by the job. The simultaneous start of this instruction with other instructions is allowed.

In contrast to Motion Control instructions, for MCL motion jobs the blending is defined in the preceding job and not in the following job.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Requirements

- The following technology objects have been configured correctly:
  - Kinematics
  - Kinematics axes
  - Interpreter
  - Interpreter program
- The kinematics is connected to the Interpreter.
- The axes interconnected with the kinematics are enabled.
- A single-axis job (e.g. "move()") is not active on any of the axes interconnected with the kinematics.

Syntax

MCL

```
ptpJtRel( <jtPos> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] [,trans := <val>]
[,blendDist := <val>] );
```

Parameters

The following table shows the parameters of the "ptpJtRel()" instruction:

Parameter	Data type	Default value	Description	
jtPos	TO_Struct_Ipr_JtPosition	-	Relative joint target position in the joint coordinate system (JCS)	
j1	LREAL	0.0	Relative target position of joint J1 in the JCS	
j2	LREAL	0.0	Relative target position of joint J2 in the JCS	
j3	LREAL	0.0	Relative target position of joint J3 in the JCS	
j4	LREAL	0.0	Relative target position of joint J4 in the JCS	
j5	LREAL	0.0	Relative target position of joint J5 in the JCS	
j6	LREAL	0.0	Relative target position of joint J6 in the JCS	
v	LREAL	-	Percentage factor for the velocity of the axis movements in relation to the respective maximum velocity of the axes ("<TO_Axis>.DynamicLimits.MaxVelocity") [1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor". With setPtpDyn() (Page 240), you can set another value modally.
			1.0 ≤ v ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			< 1.0	Not permitted
a	LREAL	-	Percentage factor for the acceleration of the axis movements in relation to the respective maximum acceleration of the axes ("<TO_Axis>.DynamicLimits.MaxAcceleration")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < a ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			≤ 0.0	Not permitted
d	LREAL	-	Percentage factor for the deceleration of the axis movements in relation to the respective maximum deceleration of the axes ("<TO_Axis>.DynamicLimits.MaxDeceleration")[1]	

[1] Modal parameter

Parameter	Data type	Default value	Description	
d	LREAL	-	-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor". With setPtpDyn() (Page 240), you can set another value modally.
			0.0 < d ≤ 100.0	The specified value is used.
			> 100.0	Not permitted
			<= 0.0	Not permitted
j	LREAL	-	Percentage factor for the jerk of the axis movements in relation to the respective maximum jerk of the axes ("<TO_Axis>.DynamicLimits.MaxJerk")[1]	
			-	The modal value is used. The modal value is initialized with "<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor". With setPtpDyn() (Page 240), you can set another value modally.
			10.0 <= j ≤ 90.0	The specified value is used.
			= 0.0	No jerk limit
			< 10.0 (except 0.0)	Not permitted
			> 90.0	Not permitted
trans	DINT	-	Motion transition to the next job[1]	
			-	The modal value is used. By default, the "trans" parameter is preset to the value "0". With setTrans() (Page 247), you can set another value modally.
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs
			If "trans" = 1 or "trans" = 2: Always polynomial blending between sPTP and path motions.	
blendDist	LREAL	-	Blending distance[1]	
			-	The modal value is used. By default, the "blendDist" parameter is preset to the value "-1.0". With setBlendDist() (Page 252), you can set another value modally.
			≥ 0.0	The default value is used.
			< 0.0	The maximum possible blending distance is used.

[1] Modal parameter

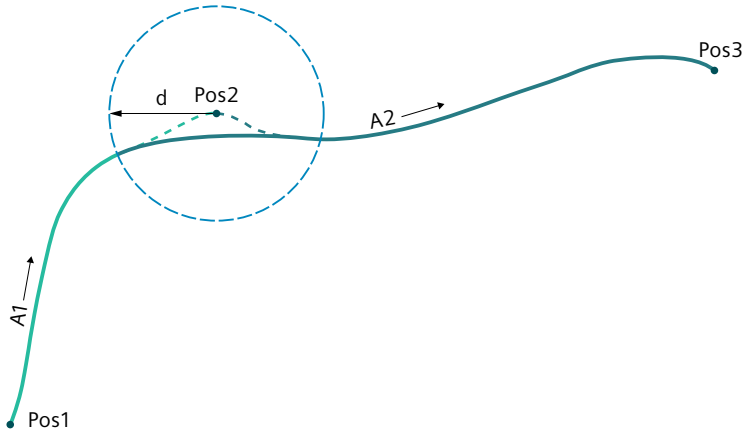
Comparable Motion Control instructions

- "MC_MoveDirectRelative": Relative movement of kinematics with synchronous "point-to-point" motion (if "CoordSystem" = 101)

Example: Positioning kinematics with sPTP motion at relative joint positions in the joint coordinate system

In the following example, a kinematics is to move to two relative joint target positions in the joint coordinate system.

The kinematics should move from standstill Position 1 with blending distance ("blendDist" = d) and at high velocity ("trans" = 2) via Position 2 (A1) to Position 3 (A2) ("trans" = 0).



MCL

```
VAR
   Pos2, Pos3 : TO_Struct_Ipr_JtPosition;
   d : LREAL;
END_VAR
// A1 - sPTP movement with blending with high velocity
// through Pos2 with coordinate Pos2
// relative to Pos1 in JCS with max. sPTP dynamics without jerk limitation
ptpJtRel( Pos2, v := 100.0, a := 100.0, d := 100.0, j := 0.0, trans := 2, blendDist := d );
// A2 - sPTP movement without blending to Pos3
// with coordinate Pos3 relative to Pos2 in JCS
// Exact stop in Pos3
ptpJtRel( Pos3, trans := 0 );
```

8.1.11 setDyn() Set dynamic defaults for path motions modally (S7-1500T)

Description

Use the MCL instruction "setDyn()" to set the modal values for the dynamic default of the path motion. If you do not predefine any other dynamic values for an MCL motion job, these modal dynamic values are used for the kinematics motion.

You can specify one or more dynamic values. The specification is possible as an absolute or percentage value in relation to the maximum dynamics.

Specify a value for at least one of the dynamic parameters.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()

Syntax

MCL

```
setDyn( [v := <value>] [,a := <value>] [,d := <value>] [,j := <value>] [,rel := <value>] );
```

Parameters

The following table shows the parameters of the instruction "setDyn()":

Parameter	Data type	Default value	Description	
v	LREAL	-	Modal value of the velocity of the path motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Path.Velocity). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the velocity
			0.0 < v ≤ 100.0	With rel := TRUE: Percentage default of the last valid maximum velocity value
			≤ 0.0	Not permitted
a	LREAL	-	Modal value of the acceleration of the path motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Path.Acceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the acceleration
			0.0 < a ≤ 100.0	With rel := TRUE: Percentage default of the last valid maximum acceleration value
			≤ 0.0	Not permitted
d	LREAL	-	Modal value of the deceleration of the path motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Path.Deceleration). If the value is not set in the program code, the initialized value is used.

[1] Optional parameter with default value

Parameter	Data type	Default value	Description	
d	LREAL	-	> 0.0	With rel := FALSE: Absolute default of the deceleration
			0.0 < d ≤ 100.0	With rel := TRUE: Percentage default of the last valid maximum deceleration value
			≤ 0.0	Not permitted
j	LREAL	-	Modal value of the jerk of the path motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Path.Jerk). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the jerk
			0.0 < j ≤ 100.0	With rel := TRUE: Percentage default of the last valid maximum jerk value
			= 0.0	No jerk limit
			< 0.0	Not permitted
rel	BOOL	FALSE	Type of specification of the modal value[1)]	
			FALSE	Absolute specification
			TRUE	Percentage specification

1) Optional parameter with default value

Examples

Set modal dynamic values:

MCL

```
setDyn( rel := FALSE,
      v := 100.0,
      a := 1000.0,
      d := 1000.0,
      j := 0.0 );
```

Set acceleration of path motion to 50% of the maximum value of the last valid velocity value:

MCL

```
setDyn( a := 50.0, rel := TRUE );
```

Set modal dynamic values for a and d:

MCL

```
setDyn( a := 10.0, d := 10.0 );
```

8.1.12 setOriDyn() Set dynamic defaults for orientation motions modally (S7-1500T)

Description

Use the MCL instruction "setOriDyn()" to set the modal values for the dynamic default of the orientation motion.

You can specify the values for one or more modal dynamic parameters. The specification is possible as an absolute value and as a relative value in percent related to the maximum orientation dynamics (see instruction "setOriDynMax()"). All dynamic values not specified in the instruction remain unchanged.

Specify a value for at least one of the parameters.

You can find more information on the dynamic defaults in the "Modal parameters (Page 179)" section.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics with orientation

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()

Syntax

MCL

```
setOriDyn( [v := <value>] [,a := <value>] [,d := <value>] [,j := <value>] [,rel := <value>]
);
```

Parameters

The following table shows the parameters of the instruction "setOriDyn()":

Parameter	Data type	Default value	Description	
v	LREAL	-	Modal value of the velocity of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Orientation.Velocity). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the velocity
			0.0 < v ≤ 100.0	With rel := TRUE: Percentage default of the last maximum valid velocity value
			≤ 0.0	Not permitted
a	LREAL	-	Modal value of the acceleration of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Orientation.Acceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the acceleration
			0.0 < a ≤ 100.0	With rel := TRUE: Percentage default of the last maximum valid acceleration value
			≤ 0.0	Not permitted
d	LREAL	-	Modal value of the deceleration of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Orientation.Deceleration). If the value is not set in the program code, the initialized value is used.

[1] Optional parameter with default value

Parameter	Data type	Default value	Description	
d	LREAL	-	> 0.0	Absolute default of the deceleration limit
			0.0 < d ≤ 100.0	With rel := TRUE: Percentage default of the last maximum valid deceleration value
			≤ 0.0	Not permitted
j	LREAL	-	Modal value of the jerk of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.Orientation.Jerk). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the jerk
			0.0 < j ≤ 100.0	With rel := TRUE: Percentage default of the last maximum valid jerk value
			= 0.0	No jerk limit
			< 0.0	Not permitted
rel	BOOL	FALSE	Type of specification of the modal value[1)]	
			FALSE	Absolute specification
			TRUE	Percentage specification

[1)] Optional parameter with default value

Example

MCL

```
// set orientation dynamic modally, absolute values
setOriDyn( rel := FALSE,
         v := 100.0,
         a := 1000.0,
         d := 1000.0,
         j := 0.0 );
// orientation movement will be done with the modal orientation dynamic
linAbs( myPos1 );
// set the value of the orientation deceleration to absolute 2000.0
setOriDynMax( d := 2000.0 );
// set modally deceleration for orientation movement
// in % related to the maximum value of deceleration for orientation movement
// all other dynamics parameters (v, a, j) are not changed
setOriDyn( rel := TRUE, d := 25.0 );
// orientation movement will be done with deceleration 500.0 - absolute value
linAbs( myPos2 );
```

8.1.13 setPtpDyn() Setting dynamic defaults for sPTP motions modally (S7-1500T)

Description

With the MCL instruction "setPtpDyn()", you can set the modal values for the dynamic default of the sPTP motions. If you do not specify any other dynamic values for an MCL motion job, these modal dynamic values are used for the kinematics sPTP motions.

You can specify the values for one or more modal dynamic parameters. Any dynamic value not specified in the instruction remains unchanged.

Specify a value for at least one of the dynamic parameters.

You can find more information on the dynamic defaults in the section "Modal parameters (Page 179)" section.

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- ptpAbs()
- ptpRel()
- ptpJtAbs()
- ptpJtRel()
- ptpAxAbs()
- ptpAxRel()

Syntax

MCL

```
setPtpDyn( [v := <value>] [,a := <value>] [,d := <value>] [,j := <value>] );
```

Parameters

The following table shows the parameters of the "setPtpDyn()" instruction:

Parameter	Data type	Default value	Description	
v	LREAL	-	Modal value of the percentage factor for the velocity of the sPTP motion related to the respective maximum velocity of the axes ("<TO_Axis>.DynamicLimits.MaxVelocity").	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.MoveDirect.VelocityFactor). If the value is not set in the program code, the initialized value is used.
			1.0 ≤ v ≤ 100.0	Percentage value of the maximum axis velocity.
			< 1.0	Not permitted.
			> 100.0	Not permitted.
a	LREAL	-	Modal value of the percentage factor for the acceleration of the sPTP motion related to the respective maximum acceleration of the axes ("<TO_Axis>.DynamicLimits.MaxAcceleration").	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.MoveDirect.AccelerationFactor). If the value is not set in the program code, the initialized value is used.
			0.0 < a ≤ 100.0	Percentage value of the maximum axis acceleration
			≤ 0.0	Not permitted.
			> 100.0	Not permitted.
d	LREAL	-	Modal value of the percentage factor for the deceleration of the sPTP motion related to the respective maximum deceleration of the axes ("<TO_Axis>.DynamicLimits.MaxDeceleration").	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.MoveDirect.DecelerationFactor). If the value is not set in the program code, the initialized value is used.
			0.0 < d ≤ 100.0	Percentage value of the maximum axis deceleration
			≤ 0.0	Not permitted.

Parameter	Data type	Default value	Description	
d	LREAL	-	> 100.0	Not permitted.
j	LREAL	-	Modal value of the percentage factor for the jerk of the sPTP motion related to the respective maximum jerk of the axes ("<TO_Axis>.DynamicLimits.MaxJerk").	
			-	The modal value remains unchanged. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.MoveDirect.JerkFactor). If the value is not set in the program code, the initialized value is used.
			10.0 ≤ j ≤ 90.0	Percentage value of the maximum axis jerk
			< 10.0 (except 0.0)	Not permitted.
			> 90.0	Not permitted.
			= 0.0	No jerk limit

Example

MCL

```
// set the modal factor of dynamics of a sPTP motion without jerk limitation
setPtpDyn( v := 10.0, a := 20.0, d := 30.0, j := 0.0 );
…
// set the modal factor of the velocity of a sPTP motion to 75.0%
setPtpDyn( v := 75.0 )
```

8.1.14 setDynMax() Set dynamic limits for path motions modally (S7-1500T)

Description

Use the MCL instruction "setDynMax()" to set the maximum limits for the modal dynamic parameters of linear and circular path motions.

You can set the maximum dynamic values for one or more modal parameters. Maximum dynamic values can be specified only in absolute values. Any dynamic value not specified in the instruction remains unchanged.

Specify a value for at least one of the parameters.

You can find more information on the dynamic defaults in the section "Modal parameters (Page 179)".

You can find more information on linear and circular path motions in the sections "Move kinematics linearly" and "Move kinematics circularly" of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()

Syntax

MCL

```
setDynMax( [v := <value>] [,a := <value>] [,d := <value>] [,j := <value>] );
```

Parameters

The following table shows the parameters of the instruction "setDynMax()":

Parameter	Data type	Default value	Description	
v	LREAL	-	Modal value of the maximum velocity of the path motion	
			-	The modal value remains unchanged. The most recent valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Path.Velocity). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of velocity limits
			≤ 0.0	Not permitted
a	LREAL	-	Modal value of the maximum acceleration of the path motion	
			-	The modal value remains unchanged. The most recent valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Path.Acceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the limits of acceleration
			≤ 0.0	Not permitted

Parameter	Data type	Default value	Description	
d	LREAL	-	Modal value of the maximum deceleration of the path motion	
			-	The modal value remains unchanged. The most recent valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Path.Deceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the deceleration limits
			≤ 0.0	Not permitted
j	LREAL	-	Modal value of the maximum jerk of the path motion	
			-	The modal value remains unchanged. The most recent valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Path.Jerk). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the jerk limits
			= 0.0	No jerk limit
			< 0.0	Not permitted

Example

MCL

```
// set the maximum value of the path velocity to absolute 10.0
setDynMax( v := 10.0 );
linAbs( Pos1, v := 8.0 ); // Path movement with velocity = 8.0
// set the maximum value of the path acceleration and deceleration
setDynMax( a := 1000.0, d := 1000.0 );
// path movement with v = 10.0, a = 1000.0 and d = 800.0
// dynamics are limited by setDynMax()- instruction
// in the following instruction v is limited to 10.0 and a is limited to 1000.0
linAbs( Pos1, v := 15.0, a := 2000.0, d := 800.0 );
```

8.1.15 setOriDynMax() Set dynamic limits for orientation motions modally (S7-1500T)

Description

Use the MCL instruction "setOriDynMax()" to set the maximum limits for the modal dynamic parameters of orientation motions.

You can specify maximum dynamic values for one or more modal dynamic parameters. The specification is possible only in absolute values. Any dynamic value not specified in the instruction remains unchanged.

Specify a value for at least one of the parameters.

You can find more information on the maximum limits in the section "Modal parameters (Page 179)".

You can find more information on orientation motions in the "Kinematics motions overview" section of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics with orientation

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()

Syntax

MCL

```
setOriDynMax( [v := <value>] [,a := <value>] [,d := <value>] [,j := <value>] );
```

Parameters

The following table shows the parameters of the instruction "setOriDynMax()":

Parameter	Data type	Default value	Description	
v	LREAL	-	Modal value of maximum velocity of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Orientation.Velocity). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the velocity limit
			≤ 0.0	Not permitted
a	LREAL	-	Modal value of the maximum acceleration of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Orientation.Acceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the acceleration limit
			≤ 0.0	Not permitted
d	LREAL	-	Modal value of the maximum deceleration of the orientation motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Orientation.Deceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the deceleration limit
			≤ 0.0	Not permitted
j	LREAL	-	Modal value of the maximum jerk of the orientation motion	

Para-meter	Data type	Default value	Description	
j	LREAL	-	-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicLimits.Orientation.Jerk). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the jerk limit
			= 0.0	No jerk limit
			< 0.0	Not permitted

Examples

MCL

```
// set max. of jerk
// max. values for velocity, acceleration and deceleration
// used from technology object data block if not set before in the program
setOriDynMax( j := 10000.0 );
```

MCL

```
setOriDynMax( v := 100.0, // set max. orientation dynamic
              a := 2000.0,
              d := 2500.0,
              j := 0.0 );
```

8.1.16 setTrans() Setting motion transition for linear, circular path, and sPTP motions modally (S7-1500T)

Description

With the MCL instruction "setTrans()", you can modally define the blending dynamics between two movements. You can choose from "Append motion", "Blend with the lower velocity of the two jobs", and "Blend with the higher velocity of the two jobs".

The last valid value of the linear, circular path, and sPTP motion specification set in the program code is used for the blending dynamics. "Append motion" is preset ("trans" = 0) by default.

You can find more information on blending in the sections "Move kinematics linearly", "Move kinematics circularly", and "Move kinematics with a synchronous "point-to-point" motion" of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()
- ptpAbs()
- ptpRel()
- ptpJtAbs()
- ptpJtRel()
- ptpAxAbs()
- ptpAxRel()

Syntax

MCL

```
setTrans( <trans> );
```

Parameters

The following table shows the parameters of the "setTrans()" instruction:

Parameter	Data type	Default value	Description	
trans	DINT	-	Modal value for the motion transition to the next instruction of the path and sPTP path motions	
			0	Append motion
			1	Blend with the lower velocity of the two jobs.
			2	Blend with the higher velocity of the two jobs

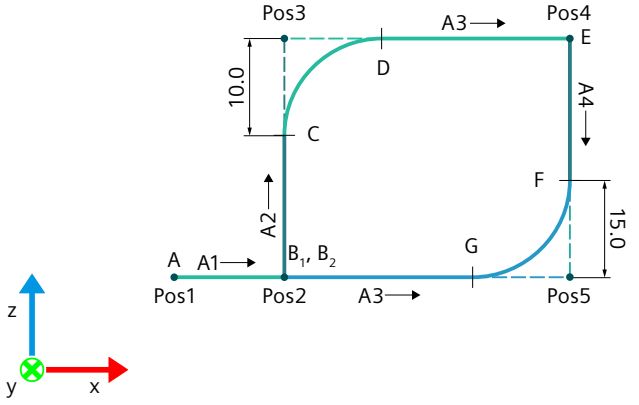
Example: Blending over motion with lower and higher velocity

In the following example, a kinematics is to move to five absolute positions with different specifications for the motion transition.

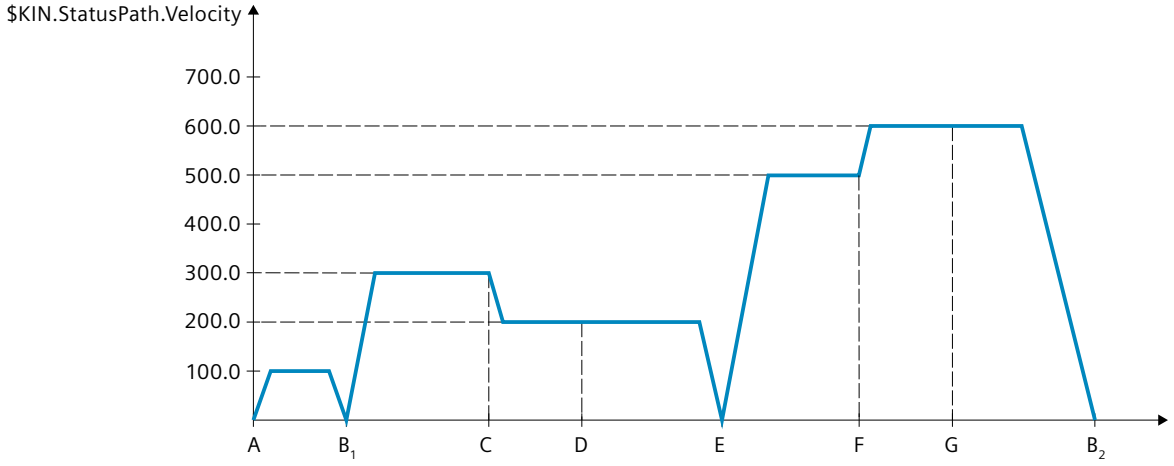
The kinematics should move from standstill from position 1 to position 2 and stop at position 2 (A1).

From position 2 the kinematics should move with blending distance 10.0 and with lower velocity via position 3 (A2) to position 4 and stop there (A3).

From position 4, the kinematics should move with an blending distance of 15.0 and at a higher velocity via position 5 (A4) to position 2 and stop at position 2 (A5).



The velocity of the kinematics motion on the different sections of the path corresponds to the following values:



MCL

```
PROGRAM main
  VAR
    Pos2, Pos3, Pos4, Pos5 : TO_Struct_Ipr_Position;
  END_VAR
// set dynamics modally
  setDyn( v := 100.0, a := 200.0, d := 200.0, j := 3000.0 );

// set blending dynamics modally to the movement without blending
  setTrans( 0 );

// A1 - Exact stop in Pos2
  linAbs( Pos2, cs := 0 ); // linear movement without blending
```

MCL

```
// A2 - Blending with lower velocity (v = 200.0) and blending distance 10.0 in Pos3
  linAbs( Pos3, v := 300.0, trans := 1, blendDist := 10.0, cs := 0 );

// A3 - Movement to Pos4 with v = 200.0. Exact stop in Pos4, no blending
  linAbs( Pos4, v := 200.0, trans := 0 );

// A4 - Blending higher velocity (v = 600.0) with blending distance 15.0 in Pos5
  linAbs( Pos5, v := 500.0, trans := 2, blendDist := 15.0, cs := 0 );

// A5 - Exact stop in Pos2, trans = 1 is ignored because this is the last command
  linAbs( Pos2, v := 600.0, trans := 1, cs := 0 );
END_PROGRAM
```

8.1.17 setBlend() Set blending mode for linear and circular path motions modally (S7-1500T)

Description

If the modal parameters for blending between two path jobs are changed, they always influence the following instructions.

The last valid mode set in the program code is used for blending. By default, the "Polynomial" mode is preset ("blend" = 2).

You can find more information on blending motions in the sections "Move kinematics linearly" and "Move kinematics circularly" of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()

Syntax

MCL

```
setBlend( <blend> );
```

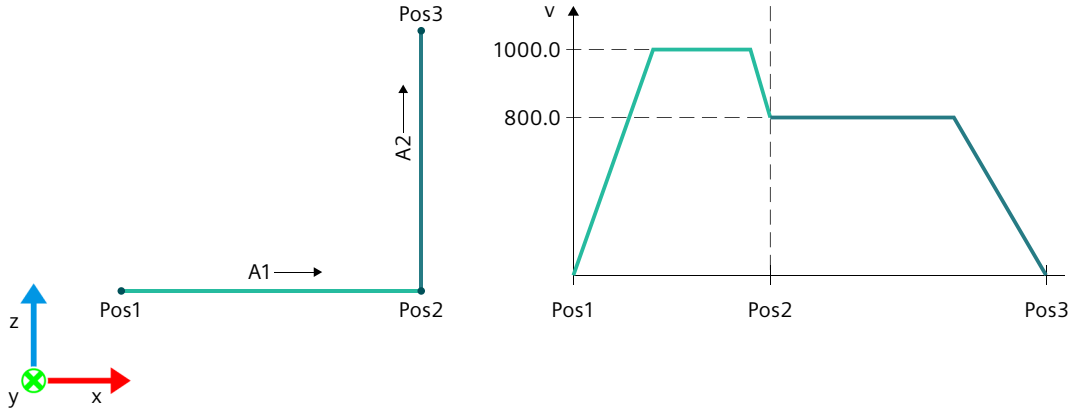
Parameters

The following table shows the parameters of the "setBlend()" instruction:

Parameter	Data type	Default value	Description	
blend	DINT	-	Modal value for blending mode	
			0	Direct Blending takes place in the path dynamics without geometric corrections. The geometry at the blending point is not changed. The kinematics travels directly through the blending point.
			1	Reserved
			2	Polynomial Geometric blending with polynomial path A polynomial path is used as a transition between two geometry elements within the blending distance (see also "setBlendDist()")

Example 1

The following example shows how the kinematics moves to two positions.



From standstill Pos1, the kinematics ("v" = 1000.0) moves with direct blending ("blend" = 0) and lower velocity ("trans" = 1, "v" = 800.0) via Pos2 (A1) to Pos3 (A3).

MCL

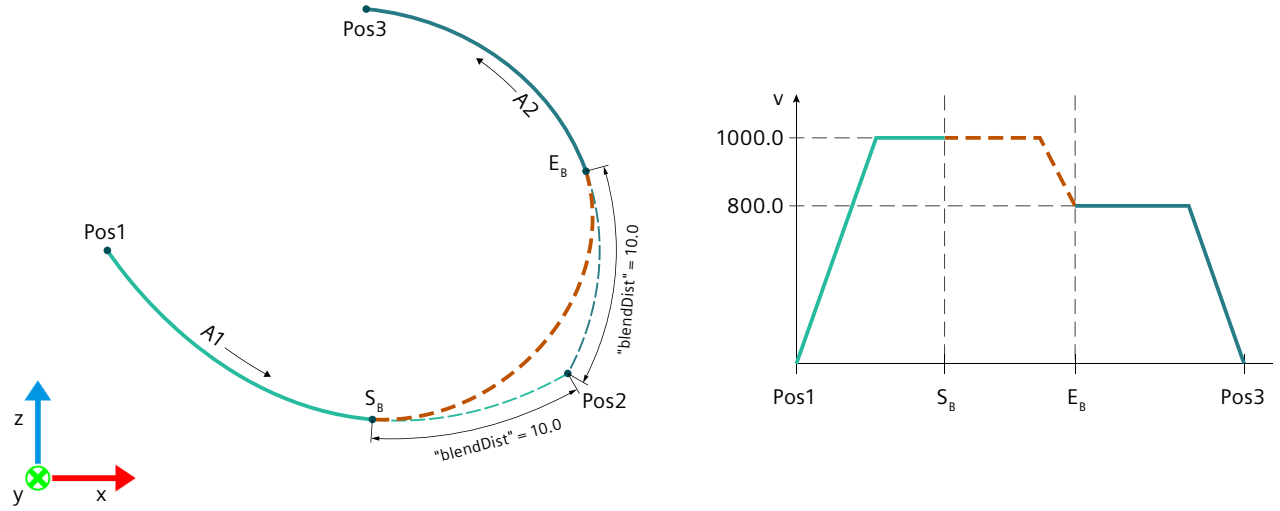
```
VAR
   Pos2, Pos3 : TO_Struct_Ipr_Position
END_VAR
setBlend( 0 ); // modal specification for direct blending
// A1 - Direct blending with low velocity (v = 800.0) in Pos2
linAbs( Pos2, v := 1000.0, trans := 1, cs := 0 );

// A2 - Direct stop in Pos3
linAbs( Pos3, v := 800.0 );
```

Example 2

The following example shows how the kinematics moves to two positions.

From standstill Pos1, the kinematics ("v" = 1000.0) moves with polynomial blending ("blend" = 2), higher velocity ("trans" = 2, "v" = 1000.0), and blending distance via Pos2 (A1) to Pos3 (A3). S_B is the start point of the blending segment, E_B is the end point of the blending segment.



S_B start blending

E_B end blending

MCL

```
VAR
   Pos2, Pos3 : TO_Struct_Ipr_Position;
END_VAR
setBlend( 2 ); // modal specification for polynomial blending
// A1 - Polinom blending with high velocity (v = 1000.0) in Pos2
circAbs( Pos2, v := 1000.0, trans := 2, blendDist := 10.0, cs := 0 );

// A2 - Direct stop in Pos3
circAbs( Pos3, v := 800.0 );
```

8.1.18 setBlendDist() Set blending distance for linear, circular path and sPTP motions modally (S7-1500T)

Description

Use the MCL instruction "setBlendDist()" to set the blending distance of linear, circular and sPTP motions modally as an absolute value.

The last valid value set in the program code is used for the blending distance. By default, the maximum blending distance is preset ("blendDist" = -1.0).

You can find more information on the maximum blending distance and the blending of motions in the sections "Kinematics motions overview", "Move kinematics linearly", "Move kinematics circularly", and "Move kinematics with a synchronous "Point-to-point" motion" of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()
- ptpAbs()
- ptpRel()
- ptpJtAbs()
- ptpJtRel()
- ptpAxAbs()
- ptpAxRel()

Syntax

MCL

```
setBlendDist( < blendDist> );
```

Parameters

The following table shows the parameters of the instruction "setBlendDist()":

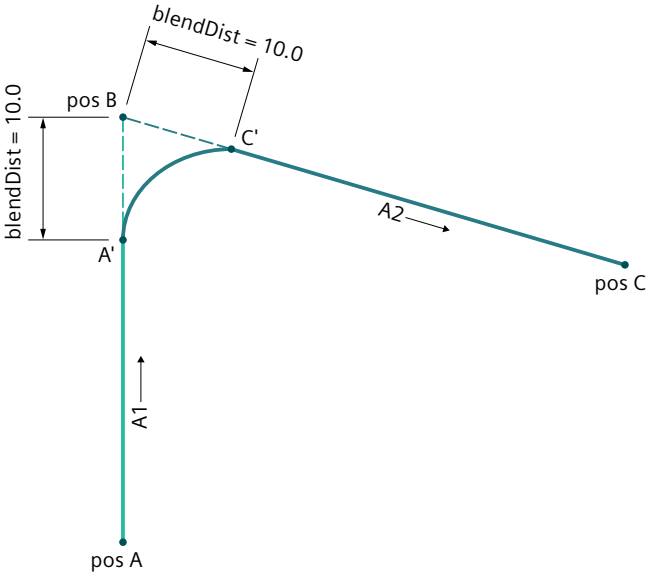
Parameter	Data type	Default value	Description	
blendDist	LREAL	-	Modal value for the blending distance (for linear path and sPTP motions)/the blending radius path (for circular path motions)	
			≥ 0.0	The default absolute value is used.
			< 0.0	The maximum possible blending distance is used, which is configured with "blendingFactor" (see also "setBlendFactor()" instruction)

Examples

The two following examples show the same motion of the kinematics with the same blending process. The difference between the examples is the specification of the blending mode and blending distance. In the first example, the blending mode is set by "setBlend()" and the blending distance is set by "setBlendDist()". In the second example, these parameters are set by the "linAbs()" path motion instruction.

With the "Geometric blending" mode, the transition of the instructions takes place within a specified blending distance. In example 1 this is realized by the "setBlendDist()" instruction and in example 2 by the "linAbs()" instruction.

The maximum blending distance is defined as the path length of the first job (A1) between the end point of the path (pos B) and the blending point A' or as the path length of the second job (A2) between the end point of the path (pos B) and the blending point C'. When the blending distance is reached, the current motion is blended with the new motion. The geometric transition is automatically applied by the system.



pos A	Starting point of kinematics
pos B	End point of the first job (A1)
pos C	End point of the second job (A2)
blend-Dist	End point of the second job (A2)

MCL (example 1)

```
setBlend( 2 ); // modally setting for polynomial blending
setBlendDist( 10.0 ); // modally setting for blending distance
linAbs( posB, trans := 1 ); // blending active (command A1)
linAbs( posC, trans := 0 ); // exact stop at command end (command A2)
```

MCL (example 2)

```
// using path-movements instruction linAbs()
// as an alternative way of setting blending mode and blending distance
linAbs( posB, trans := 1, blend := 2, blendDist := 10.0 ); // (command A1)
linAbs( posC ); // (command A2)
```

8.1.19 setBlendFactor() Set blending distance for linear, circular path and sPTP motions modally (S7-1500T)

Description

Use the MCL instruction "setBlendFactor()" to modally set the factor of the maximum blending distance of the kinematics for linear and circular path motions as well as sPTP motions.

This factor allows a blending distance greater than half the path length of the sPTP motion.

The input and output blending distance at the blending point remains unchanged.

This factor is only taken into account if the blending distance for the motion is set to the maximum possible blending distance ("blendDist" = -1.0, see also "setBlendDist()" instruction).

The last valid value set in the program code is used for the factor. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Job sequence" (<TO_Kinematics>.Transition.FactorBlendingLength). If the value is not set in the program code, the initialized value is used.

You can find more information on the blending of motions and the blending distance factor in the sections "Move kinematics linearly" and "Move kinematics circularly" of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()
- ptpAbs()
- ptpRel()
- ptpJtAbs()
- ptpJtRel()
- ptpAxAbs()
- ptpAxRel()

Syntax

MCL

```
setBlendFactor( <blendFactor> );
```

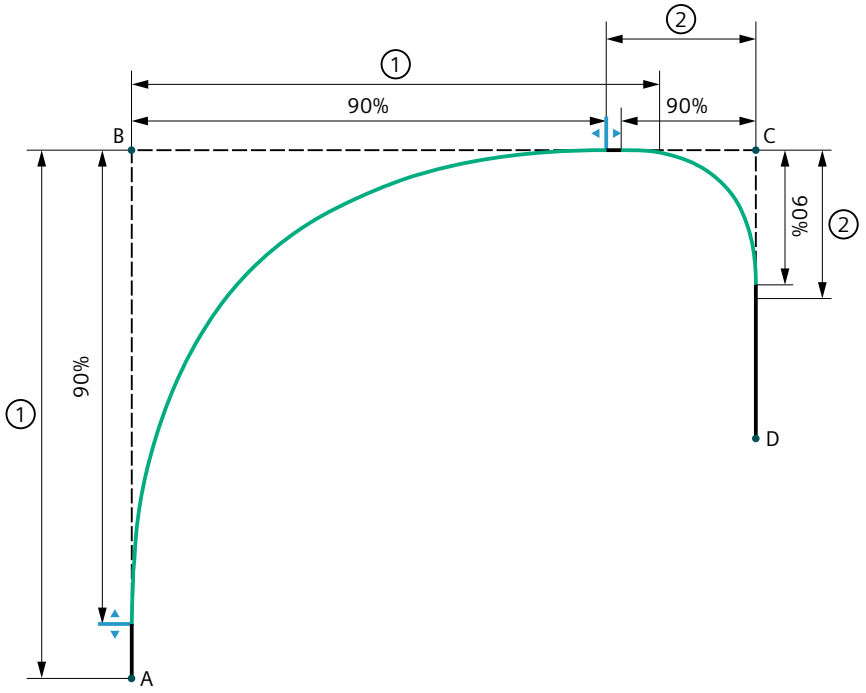
Parameters

The following table shows the parameters of the instruction "setBlendFactor()":

Parameter	Data type	Default value	Description	
blendFactor	LREAL	-	Modal value of the blending factor	
			= 0.0	No blending
			0.0 < blendFactor < 100.0	Percentage default of the factor for the motion blending.
			= 100.0	Maximum blending distance

Example 1

The following example shows a linear motion with a factor of 90.0% of the maximum possible blending distance. This factor is taken into account when the maximum possible blending distance is set for the motion. In this example, the maximum possible blending distance is set modally by the "setBlendDist()" instruction.



① max. blending distance motion job BC

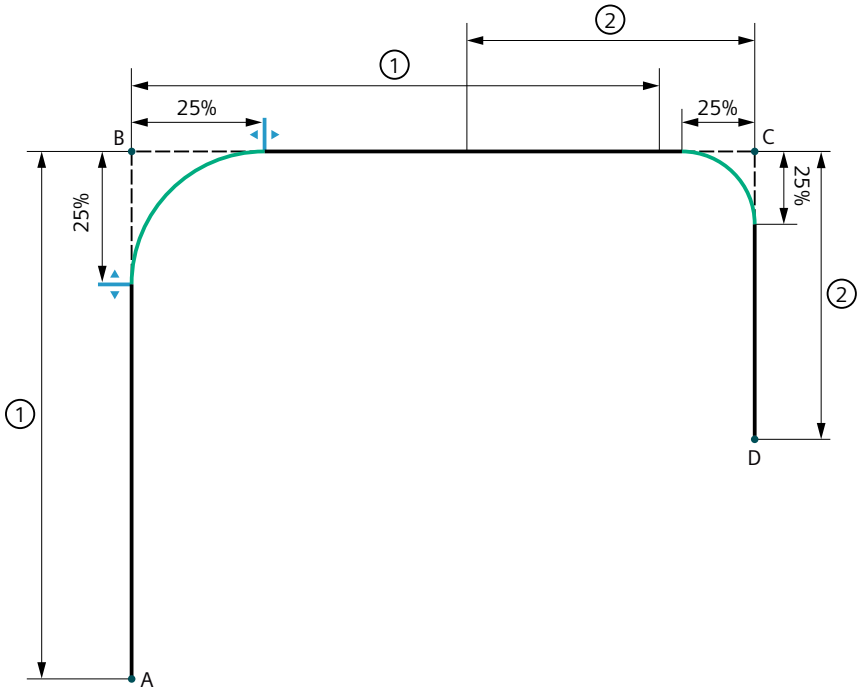
② max. blending distance motion job CD

MCL

```
setBlend( 2 ); // modal specification for polynomial blending
setBlendFactor( 90.0 ); // 90.0 % of path length for blending
setBlendDist( -1.0 ); // set max. blending distance, here 90.0%
linAbs( posB, v := 200.0, trans := 1 ); // path-movement with blending
linAbs( posC, v := 300.0 ); // path-movement with blending
linAbs( posD, v := 400.0, trans := 0 ); // path-movement without blending
```

Example 2

The following example shows a linear motion with the factor 25.0% of the maximum possible blending distance. This factor is taken into account when the maximum possible blending distance is set for the motion. In this example, the maximum possible blending distance is set modally by the "linAbs()" instruction (parameter "blendDist" = -1.0).



① max. blending distance motion job BC

② max. blending distance motion job CD

MCL

```
setBlend( 2 ); // modal specification for polynomial blending
setBlendFactor( 25.0 ); // 25.0 % of path length for blending
linAbs( posB, v := 200.0, trans := 1, blendDist := -1.0 ); // path-movement with blending
linAbs( posC, v := 300.0 ); // path-movement with blending
linAbs( posD, v := 400.0, trans := 0 ); // path-movement without blending
```

8.1.20 setPlane() Set main plane of circle path for circular path motions modally (S7-1500T)

Description

Use the MCL instruction "setPlane()" to modally set the 2D main plane for a circular path motion.

This setting of the main plane is relevant if the circle path is defined for absolute or relative circular path motions in one of the following ways:

- Definition by circle center and angle in the main plane ("mode" = 1).
- Definition by circle radius and end point in the main plane ("mode" = 2).

If an absolute or relative circular path motion in a 3D kinematics is defined via intermediate and end point ("mode" = 0), the setting of the main plane ("plane") is ignored.

You can find more information on the absolute and relative circular path motions in the sections "circAbs() Position kinematics absolutely with circular path motion (Page 194)" and "circRel(): Relative positioning of kinematics with circular path motion (Page 201)".

For the main plane of a circular path motion, the last valid value set in the program code is used. By default, the main plane XZ is preset ("plane" = 0).

You can find more information on the main plane of a circular path motion in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- circAbs()
- circRel()

Syntax

MCL

```
setPlane( <plane> );
```

Parameters

The following table shows the parameters of the instruction "setPlane()":

Parameter	Data type	Default value	Description	
plane	DINT	-	Modal value for the main plane of the circle path	
			0	XZ plane
			1	YZ plane
			2	XY plane

Example 1

In the following example, the MCL instruction "setPlane()" modally sets the main plane for a circular path motion to XZ.

MCL

```
setPlane( 0 ); // modal specification main plane XZ

// absolute circular motion in main plane XZ
circAbs( endPos, auxPos := centerPoint, mode := 1, blend := 2 );
```

Example 2

In the following example, the MCL instruction "setPlane()" modally sets the main plane for a circular path motion to XY.

MCL

```
setPlane( 2 );


// relative circular motion in main plane XY

circRel( endPos, auxPos := centerPoint, mode := 2,
         cDir := 3, blend := 2 );
```

Example 3

In the following example, the main plane for circular path motion is not relevant because the "mode" parameter in the path motion instruction "circAbs()" is set to 0 ("mode" = 0, path motion in a 3D kinematics defined via intermediate and end point).

MCL

```
setPlane( 1 ); // modal specification main plane YZ

// absolute circular motion in 3D (parameter "mode" = 0)
circAbs( endPos, auxPos := centerPoint, mode := 0, blend := 2 );
```

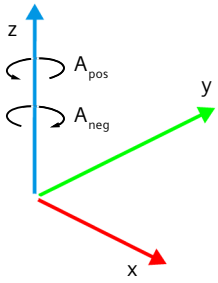
8.1.21 setOriDirA() Set direction of motion of Cartesian orientation A for linear, circular path and sPTP motions modally (S7-1500T)

Description

Use the MCL instruction "setOriDirA()" to modally set the specification of the direction of motion of the Cartesian orientation by A. This specification is only relevant for 2D/3D kinematics with orientation A in Cartesian orientation and for the linear, circle path and sPtP motion instructions with absolute position specification.

This instruction can be used to set the direction of the motion to the positive direction, the negative direction ("Three finger rule") or the motion on the shortest path.

For the specification of the direction of motion of the Cartesian orientation around A for linear, circle path and sPTP motion, the last valid value set in the program code is used. By default, the shortest path of the motion is preset ("oDirA" = 3).



You can find more information on orientation motions in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics 2D/3D with orientation A in Cartesian orientation and modulo functionality

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- circAbs()
- ptpAbs()

Syntax

MCL

```
setOriDirA( <oDirA> );
```

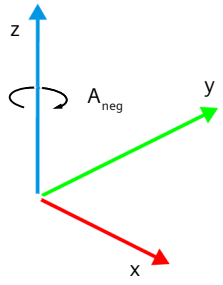
Parameters

The following table shows the parameters of the instruction "setOriDirA()":

Parameter	Data type	Default value	Description	
oDirA	DINT	-	Modal value for the direction of motion of the Cartesian orientation A	
			1	Positive direction
			2	Negative direction
			3	Shortest distance
			If the target position can be reached via two paths of equal length, the motion is in positive direction	

Example 1

In the following example, the "setOriDirA()" instruction sets the negative direction of Cartesian orientation A (around the Z axis) for the orientation motion:

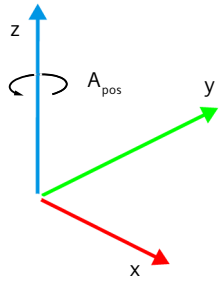


MCL

```
// set the modal value of the direction of the Cartesian orientation
// around A to 2 (negative direction)
setOriDirA( 2 );
ptpAbs( myPos1, posMode := 1, cs := 0 );
ptpAbs( myPos2, posMode := 1, cs := 0 );
```

Example 2

In the following example, the "setOriDirA()" instruction sets the positive direction of the Cartesian orientation A (around the Z axis) for the orientation motion:



MCL

```
// set the modal value of the direction of the Cartesian orientation
// around A to 1 (positive direction)
setOriDirA( 1 );
linAbs( myPos1, cs := 1, trans := 1, blend := 2 );
linAbs( myPos2 );
```

8.1.22 setTurnJoint() Setting target joint position ranges for sPTP motions modally (S7-1500T)

Description

With the MCL instruction "setTurnJoint()", you can set the position range of each joint modally. These settings are only relevant for sPTP commands with Cartesian target and joint coordinate system kinematics.

If you do not pre-define any other dynamic values for an MCL motion job, these modal dynamic values are used for the kinematics motion.

All parameters for this instruction are optional. You can specify the values for one or more modal parameters. Any value not specified in the instruction remains unchanged. Specify a value for at least one of the parameters.

The last valid value specified in the program code is used for the target joint position ranges of sPTP motions. By default, each joint is preset to the type "Shortest distance" (parameter "tj1".. "tj6" = 0).

You can find more information in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Joint coordinate system kinematics with more than 4 axes

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- ptpAbs()
- ptpRel()

The modal values for the target joint position range can alternatively be programmed with these MCL commands.

Syntax

MCL

```
setTurnJoint( [,tj1 := <val>] [,tj2 := <val>] [,tj3 :=<val>] [,tj4 := <val>]
[,tj5 := <val>] [,tj6 :=<val>] );
```

Parameters

The following table shows the parameters of the "setTurnJoint()" instruction:

Parameter	Data type	Default value	Description	
tj1, ..., tj6	DINT	-	Modal value for the target joint position range of joints J1, ..., J6	
			-	The modal value remains unchanged. By default, the "tj1",..., "tj6" parameter is preset to the value "0".
			m (m < 0)	-180° + m * 360° ≤ Position < 180° + m * 360°
			...	...
			-2	-900° ≤ Position < -540°
			-1	-540° ≤ Position < -180°
			0	Shortest distance
			1	-180° ≤ Position < 180°
			2	180° ≤ Position < 540°
			3	540° ≤ Position < 900°
			...	...
			n (n > 0)	-180° + (n - 1) * 360° ≤ Position < 180° + (n - 1) * 360°

Example

MCL

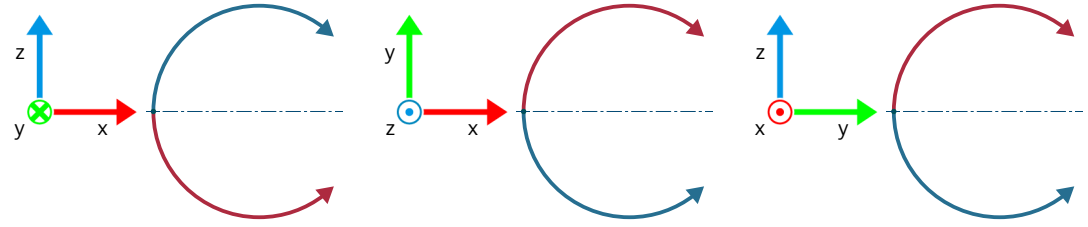
```
// set the modal value of the turn joint position range of joint 1 to -2
// (-900 <= range < -540)
setTurnJoint( tj1 := -2 );
// set the modal values of the turn joint position range of joint 2 to 1
// (-180 <= range < 180) and joint 4 to 2 (180 <= range < 540)
setTurnJoint( tj2 := 1, tj4 := 2 );
// set the modal values of the turn joint position range of joint 5 to -11
// (-4140 <= range < -3780)
setTurnJoint( tj5 := -11 );
```

8.1.23 setCircDir() Set orientation of circle path for circular path motions modally (S7-1500T)

Description

Use the MCL instruction "setCircDir()" to modally set the direction (orientation) of an absolute or relative circular orbital motion in the main plane. This direction can be positive or negative.

The following image shows the direction of a circular orbital motion in the different 2D main planes:



Positive direction of rotation

Negative direction of rotation

The direction is relevant for circular 2D path motions only if the motion is defined as follows:

- Via circle center and angle ("mode" = 1) in a main plane or
- Via circle radius and endpoint ("mode" = 2) in a main plane.

If an absolute or relative circular path motion of a 3D kinematics is defined by intermediate and end point ("mode" = 0), the default direction for this motion ("cDir") is ignored.

For the direction of a circular path motion, the last valid value set in the program code is used. By default, the shortest positive circle segment (positive direction of rotation) of the kinematics is preset ("cDir" = 0).

You can find more information on circular path motions, which are defined by circle center and angle depending on direction ("cDir") on a main plane ("mode" = 1) and on circular path motions, which is defined by the radius of the circle and the end point depending on the direction ("cDir") on a main plane ("mode" = 2), in the "S7-1500T Kinematics functions (Page 10)" documentation.

You can find more information on the "mode" parameter of a circular path motion in the sections "circAbs() Position kinematics absolutely with circular path motion (Page 194)" and "circRel(): Relative positioning of kinematics with circular path motion (Page 201)". You can find more information on the setting the main plane in the section. "setPlane() Set main plane of circle path for circular path motions modally (Page 258)".

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- circAbs()
- circRel()

Syntax

MCL

```
setCircDir( <cDir> );
```

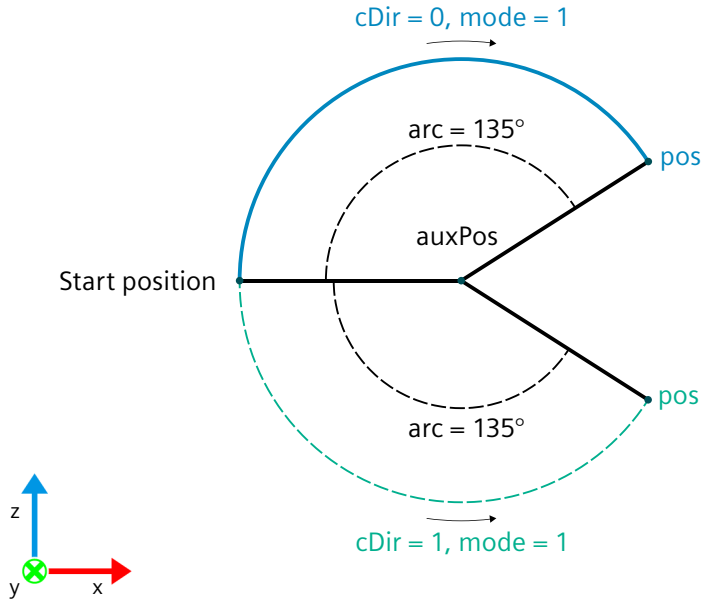
Parameters

The following table shows the parameters of the instruction "setCircDir()":

Parameter	Data type	Default value	Description	
cDir	DINT	-	Modal value for the orientation of the circle path	
			0	Positive direction of rotation ("mode" = 1)
				Shorter positive circle segment ("mode" = 2)
			1	Negative direction of rotation ("mode" = 1)
				Shorter negative circle segment ("mode" = 2)
			2	Longer positive circle segment ("mode" = 2)
			3	Longer negative circle segment ("mode" = 2)

Example 1

In the following example, the direction of the circular path motion to a positive direction of rotation in the main plane XZ is set modally by the MCL instruction "setCircDir()":



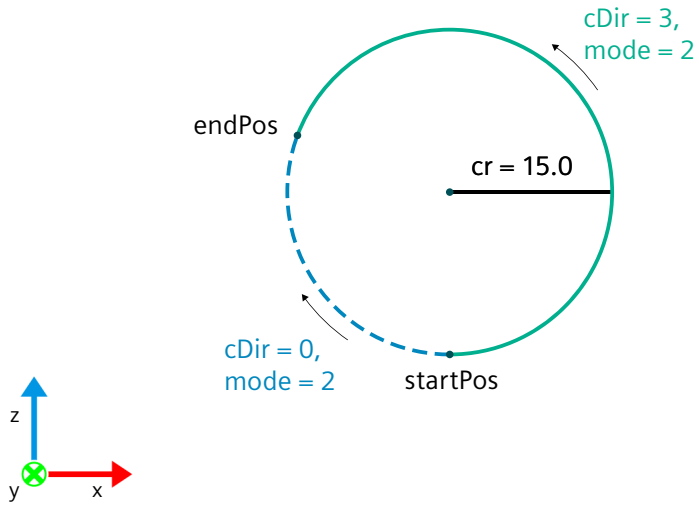
MCL

```
setPlane( 0 ); // modal specification XZ main plane
setCircDir( 0 ); // modal specification positive rotation direction

// circular path motion with interpolation over circle center point and angle ("mode" = 1)
// with positive rotation direction in XZ main plane
circAbs( posOri, auxPos := centerPoint, mode := 1, arc := 135.0 );
```

Example 2

In the following example, the direction of the circular path motion is set to the longer negative circle segment. The direction is set by the MCL instruction "circAbs()" only for this specification:



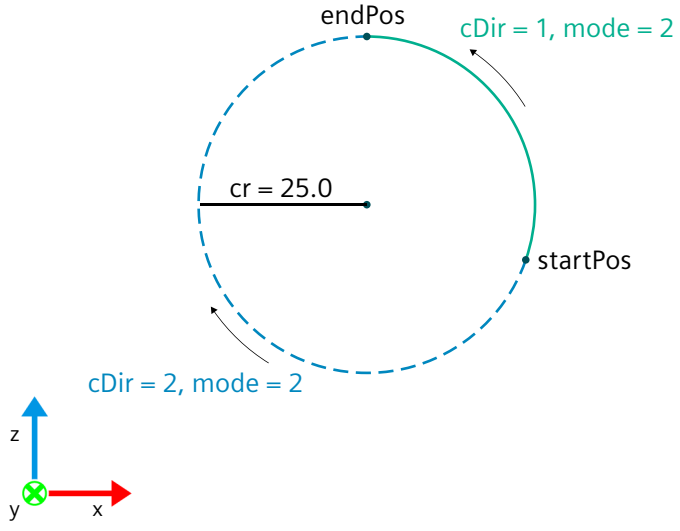
MCL

```
setPlane( 0 ); // modal specification XZ main plane

// circular motion with interpolation over circle radius and endpoint ("mode" = 2)
// with longer negative circle segment ("cDir" = 3) in XZ plane
circAbs( endPos, mode := 2, cr := 15.0, plane := 0, cDir := 3 );
```

Example 3

In the following example, the direction of the circular path motion is set to the shorter negative circle segment ("cDir" = 1). The direction is set modally by the MCL instruction "setCircDir()":



MCL

```
setPlane( 0 ); // modal specification main plane XZ
setCircDir( 1 ); // shorter negative circle segment

// circular motion with interpolation over radius ("cr" = 25.0) and endpoint ("mode" = 2)
// with shorter negative circle segment ("cDir" = 1) in XZ plane
circAbs( endPos, mode := 2, cr := 25.0 );
```

Example 4

In the following example, the direction of the circular path motion is not relevant because the "mode" parameter in the path motion job "circAbs()" is set to 0 ("mode" = 0). The circular path motion in a 3D kinematics is defined by intermediate and end point.

MCL

```
setPlane( 0 ); // modal specification main plane XZ
setCircDir( 2 ); // positive circle direction for 2D-circle - not relevant for next command

// circular motion in 3D via intermediate and endpoint
circAbs( endPos, auxPos := centerPoint, mode := 0, blend := 2 );
```

8.1.24 setLc() Setting target joint position space for sPTP motions modally (S7-1500T)

Description

With the MCL instruction "setLc()", you can modally set the target joint position space for the sPTP motion. The joint position space is set depending on the respective kinematics type.

The required "lc" parameter is interpreted as a bit field. 6 axes ($A1, $A2, $A3, $A4, $A5, $A6) correspond to the first 6 bits (bit 0 ... bit 5). The meaning of the bits depends on the type of kinematics. You can find more information on the kinematics types in the "Kinematics types" section of the "S7-1500T Kinematics functions (Page 10)" documentation.

The last valid value set in the program code is used for the target joint position space. By default, "Keep current joint position" is preset ("lc" = 16#FFFF_FFFF).

You can find more information on sPTP motion in the section "Move kinematics with a synchronous "point-to-point" motion" of the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- ptpAbs()
- ptpRel()

Alternatively, the target joint position space can be set modally with these MCL instructions.

Syntax

MCL

```
setLc( <lc> );
```

Parameters

The following table shows the parameters of the "setLc()" instruction:

Parameter	Data type	Default value	Description		
lc	DWORD	-	Modal value of the target joint position space		
			0	Negative connection constellation	
			16#FFFF_FFFF	Keep current joint position	
			Bit 0	Angle α1 of axis A1 in the front/rear area (standard area/overhead area)	
				0	The zero point of the FCS is located in the front area (standard area) of the joint position lines for axis A1 α1 = arctan(yFCS/xFCS)
					For 6-axis articulated arm with central hand (KinPlus): The wrist is located in the front area (standard area) of the joint position lines for joint J1
				1	The zero point of the FCS is located in the rear area (overhead area) of the joint position lines for axis A1 α1 = -arctan(yFCS/xFCS)
					For 6-axis articulated arm with central hand (KinPlus): The wrist is located in the rear area (overhead area) of the joint position lines for joint J1
			Bit 1	Angle α2 of axis A2 positive/negative taking into consideration the mechanical axis coupling	
				0	α2 positive
				1	α2 negative
			Bit 2	Angle α3 of axis A3 positive/negative taking into consideration the mechanical axis coupling	
				0	α3 positive

Parameter	Data type	Default value	Description		
lc	DWORD	-	Bit 2	0	For 6-axis articulated arm with central hand (KinPlus): The wrist is located on, in the standard area above, or in the overhead area below the joint position lines for joint J3
				1	α3 negative
					For 6-axis articulated arm with central hand (KinPlus): The wrist is located in the standard area below or in the overhead above the joint position lines for joint J3
			Bit 3	Angle α4 of axis A4 positive/negative taking into consideration the mechanical axis coupling	
				0	α4 negative
				1	α4 negative
			Bit 4	Geometric joint position joint J5 (KinPlus)	
				0	α5 positive
				1	α5 negative or α5 = 0°
			Bit 5	Geometric joint position joint J6 (KinPlus)	
				0	α6 positive
				1	α6 negative
			Bit 6 ... 31	Not relevant	

Example

The following example shows how to set the target joint position space:

MCL

```
…
// set target joint position space modally
setLc( 0 ); // negative link constellation

…
// set target joint position space modally
setLc( 16#3E ); // "front area" for $A1, "negative angle" for $A2...$A6

…
// set target joint position space modally
setLc( 16#FFFF_FFFF ); // keep current link constellation

…
//set target joint position in MCL instruction
ptpAbs( myPos4, posMode := 1, lc := 16#34, cs := 0 );
END_PROGRAM
```

8.1.25 setDynAdapt() Set dynamic adaptation for path motions modally (S7-1500T)

Description

Use the MCL instruction "setDynAdapt()" to set the mode for the dynamic adaptation modal for planning the dynamics considering the dynamic limits of the kinematics axes.

The dynamic adaptation limits the path dynamics to the axes dynamics. When dynamic adaptation is activated, a velocity profile is calculated for the motion of the kinematics which takes into account the following:

- Dynamic specifications or dynamic defaults and dynamic limits of the kinematics motion
- Maximum velocity, maximum acceleration and maximum deceleration of the kinematics axes.

In addition, the dynamic defaults and dynamic limits for velocity, acceleration and deceleration of the orientation motion are taken into account.

The last valid value set in the program code is used for the dynamic adaptation mode. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Kinematics>.DynamicDefaults.DynamicAdaption). If the value is not set in the program code, the initialized value is used.

You can find more information on the different modes of dynamic adaptation in the "S7-1500T Kinematics functions (Page 10)" documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()

Syntax

MCL

```
setDynAdapt( <da> );
```

Parameters

The following table shows the parameters of the instruction "setDynAdapt()":

Parameter	Data type	Default value	Description	
da	DINT	-	Modal value for dynamic adaptation	
			0	Without dynamic adaptation
			1	Limit with segmentation of path motions
			2	Limit without segmentation of path motions

Example

The following example shows how to activate and deactivate the dynamic adaptation by setting the MCL instruction "setDynAdapt()" modally.

MCL

```
// activation dynamic adaption modally - dynamic adaption with path segmentation
setDynAdapt( 1 );
linAbs( myPos1 ); // linear movement to myPos1
linRel( myPos2, v := 200.0, trans := 0 ); // linear movement to myPos2
setDynAdapt( 0 ); // deactivation dynamic adaption modally
circAbs( myPos3, auxPos := ( x := 220.0, y := 0.0, z := 250.0,
                             a := 0.0, b := 0.0, c := 0.0 ) );
linAbs( myPos1, cs := 1, trans := 1, blend := 2 );
```

8.2 Coordinate systems (S7-1500T)

8.2.1 defOcs() Redefine object coordinate systems (S7-1500T)

Description

Use the MCL instruction "defOcs()" to define the position of an object coordinate system (OCS) in relation to the world coordinate system (WCS).

The "defOcs()" job is added to the queue of the Interpreter job sequence and is therefore effective only for subsequent motion jobs.

The following variables of the technology object data block of the Kinematics technology object contain the current coordinates of the object coordinate systems:

- <TO>.StatusOcsFrame[1..3].x
- <TO>.StatusOcsFrame[1..3].y
- <TO>.StatusOcsFrame[1..3].z
- <TO>.StatusOcsFrame[1..3].a
- <TO>.StatusOcsFrame[1..3].b
- <TO>.StatusOcsFrame[1..3].c

You can find more information in the "S7-1500 Kinematics functions" (Page 10) documentation.

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.

Syntax

MCL

```
defOcs( <ocs>, <frame> );
```

Parameters

The following table shows the parameters of the MCL instruction "defOcs()":

Parameters	Data type	Default value	Description	
ocs	DINT	-	Object coordinate system	
			1	OCS1
			2	OCS2
			3	OCS3
frame	TO_Struct_Ipr_Frame	-	Coordinates in relation to the WCS	

Comparable Motion Control instructions

"MC_SetOcsFrame": Redefine object coordinate systems

8.2.2 defTool() Redefine tool (S7-1500T)

Description

Use the MCL instruction "defTool()" to redefine the tool frame of a tool. The configured start values in "<TO_Kinematics>.Tool[1..3]" are not overwritten.

A "defTool()" job interrupts the program preparation. Program preparation is restarted after the "defTool()" job has been completed.

Configure the coordinates depending on the kinematics type used.

Kinematics type		Configurable coordinates	Coordinates predefined with "0.0"
2D	without orientation	x, z	y, A, B, C
	with orientation	z, A	x, y, B, C
3D	without orientation	x, y, z	A, B, C
	with orientation	x, y, z, A	B, C
	with 2 orientations	x, y, z, B	A, C
	with 3 orientations	x, y, z, A, B, C	-
	with central hand		

The following variables of the technology object data block of the connected kinematics contain the coordinates of tools 1 to 3:

- <TO>.StatusTool.Frame[1..3].x
- <TO>.StatusTool.Frame[1..3].y
- <TO>.StatusTool.Frame[1..3].z
- <TO>.StatusTool.Frame[1..3].a
- <TO>.StatusTool.Frame[1..3].b
- <TO>.StatusTool.Frame[1..3].c

You can find more information in the "S7-1500 Kinematics functions" (Page 10) documentation.

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.
- No single-axis job (e.g. "move()") is active on any of the axes connected to the kinematics.
- The kinematics is at a standstill.

Syntax

MCL

```
defTool( <tool>, <frame> );
```

Parameters

The following table shows the parameters of the MCL instruction "defTool()":

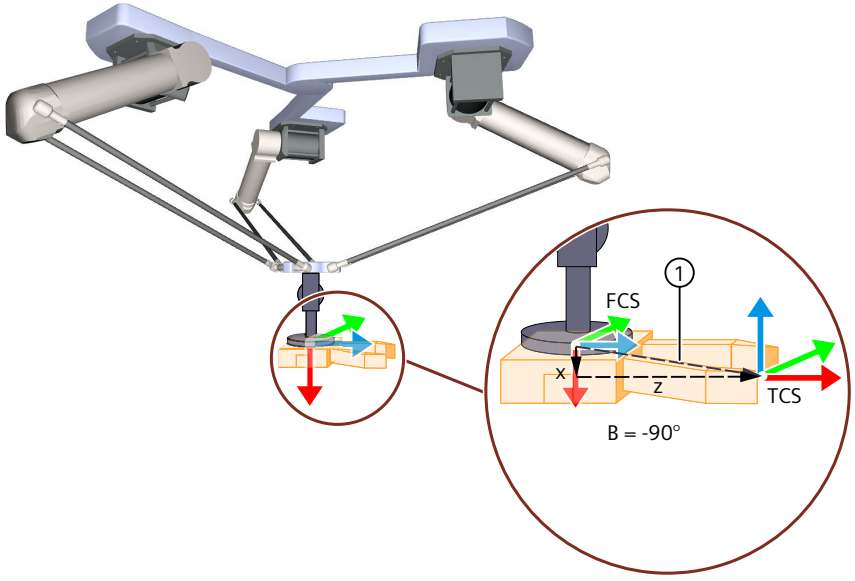
Parameters	Data type	Default value	Description	
tool	DINT	-	Number of the tool for which the tool frame is to be defined.	
			1	Tool 1
			2	Tool 2
			3	Tool 3
frame	TO_Struct_Ipr_Frame	-	Coordinates in relation to the FCS	

Comparable Motion Control instructions

"MC_DefineTool": Redefine tool

Example: Define tool frame

In the following program example, a tool frame "myToolFrame" is defined for Tool 1 on a kinematics "Deltapicker 3D with 2 orientations A, B". The tool is rotated by B = -90° in relation to the FCS and is shifted in direction x and z.



① Tool frame

MCL

```
VAR
    //Declare frame variable
    myToolFrame : TO_Struct_Ipr_Frame;
END_VAR
//x coordinate of tool frame
myToolFrame.x := 20.0;
//z coordinate of tool frame
myToolFrame.z := 100.0;
//b coordinate of tool frame
myToolFrame.b := -90.0;
//Define tool frame
defTool( 1, myToolFrame );
```

In the technology object data block of the connected kinematics, the coordinates of Tool 1 are displayed as follows:

- <TO_Kinematics>.StatusTool.Frame[1].x := 20.0
- <TO_Kinematics>.StatusTool.Frame[1].y := 0.0
- <TO_Kinematics>.StatusTool.Frame[1].z := 100.0
- <TO_Kinematics>.StatusTool.Frame[1].a := 0.0
- <TO_Kinematics>.StatusTool.Frame[1].b := -90.0
- <TO_Kinematics>.StatusTool.Frame[1].c := 0.0

8.2.3 trackIn() Start conveyor tracking (S7-1500T)

Description

Use the MCL instruction "trackIn()" to start conveyor tracking. For this, an OCS is assigned to a conductive technology object representing the conveyor belt. The OCS is assigned to an object on the conveyor using the OCS frame ("origin") and the "initPos". The OCS is then tracked with the object in x direction.

With the next kinematics motion job with target position in this OCS, the kinematics moves to the specified position in OCS and couples with the conveyor at the position.

The variable "<TO>.StatusConveyor[1..3].TrackingState" of the technology object data block of the Kinematics technology object contains the status of the conveyor tracking.

You can find more information in the "S7-1500 Kinematics functions" (Page 10) documentation.

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.
- No single-axis job (e.g. "move()") is active on any of the axes connected to the kinematics.

Syntax

MCL

```
trackIn( <axis>, <origin> [,initPos := <val>] [,ocs := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "trackIn()":

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Leading-value-capable technology object to which the OCS is coupled: Leading-value-capable technology objects are: • Positioning axis • Synchronous axis • External encoder • Leading axis proxy	
origin	TO_Struct_Ipr_Frame	-	Frame for the OCS reference position on the conveyor	
initPos	TO_Struct_Ipr_Frame	0.0	"initPos.x" contains the differential value on the belt position to determine the tracked position of the OCS in relation to the OCS reference position Permitted values: • "initPos.x" <=> 0.0 • "initPos.y" = 0.0 • "initPos.z" = 0.0 • "initPos.a" = 0.0 • "initPos.b" = 0.0 • "initPos.c" = 0.0	
ocs	DINT	1	Number of the tracked OCS	
			1	OCS1
			2	OCS2
			3	OCS3

Comparable Motion Control instructions

"MC_TrackConveyorBelt": Start conveyor tracking

Example

The kinematics follows a product on the conveyor belt with the conveyor tracking. At the specified position, the gripper picks up the product and places it on a product pallet.

For the example, the following requirements are met and not shown in the program code:

- The conductive technology object representing the conveyor belt ("Belt") has been mapped.
- The product detection and the calculation of the "initPos" are performed in the user program. The result of the product detection is made available via the following two mapping variables:
  - "productDetected" product detection completed (BOOL)
  - "initObjectPos" contains the differential value on the belt position to determine the tracked position of the OCS in relation to the OCS reference position (LREAL)

The conveyor belt is moved with a "move()" job. With a "trackIn()" job, OCS1 is assigned to the leading value capable technology object "Belt", which represents the conveyor belt. The status of conveyor tracking ("TrackingState") changes from 0 to 1.

With a "linAbs()" job, the kinematics is moved to the position specified in the OCS1. When the kinematics moves to the object position, the status of conveyor tracking changes ("TrackingState") from 1 to 2. When the kinematics follows the object position, the status of conveyor tracking changes from 2 to 3. The kinematics is then moved to the product with another "linAbs()" job. The product is picked up at the target position.

The kinematics is moved with a "linAbs()" job in the WCS and the conveyor tracking is ended in the accompanying OCS1. When the kinematics moves to the position in WCS, the "TrackingState" changes from 3 to 4. With 2 further motion commands, the kinematics is moved to the placement position on the pallet in OCS2. Once the product has been placed on the pallet, the gripper opens and the kinematics is moved to the wait position in the WCS.

MCL

MCL

```
  //move conveyor belt
  move(Belt, 100.0 );
  // wait for measuring result (product detected)
  waitEvent( productDetected = TRUE );
  //start conveyor tracking
  trackIn( Belt, (x:=50.0, z:=0.0), initPos := (x:=initObjectPos) );
  //syncronize TCP to tracked OCS
  linAbs( (x:=0.0, z:=200.0), cs := 1 );
  //movement inside tracked OCS
  linAbs( (x:=0.0, z:=50.0), trans := 1, blendDist := 20.0, cs := 1 );
  //grip product (activate gripper)
  writeVar( gripper , TRUE);
  //desynchronize TCP from tracked OCS (to WCS)
  linAbs( (z:=200.0), cs := 0 );
  linAbs( posAbovePlace, cs := 2 );
  linAbs( posPlace, cs := 2 );
  //release product (deactivate gripper)
  writeVar( gripper, FALSE);
  //move back to wait position
  linAbs( (x:=0.0, y:=0.0, z:=0.0, a:=0.0, b:=0.0, c:=0.0), trans := 1, blendDist := 50.0
);
```

8.2.4 setCs() Set reference coordinate system for linear, circular path and sPTP motions modally (S7-1500T)

Description

Use the MCL instruction "setCs()" to set the reference coordinate system for kinematics motions.

You can find more information in the "S7-1500 Kinematics functions" (Page 10) documentation.

Applies to

- Kinematics

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()
- ptpAbs()
- ptpRel()

Syntax

MCL

```
setCs( <cs> );
```

Parameters

The following table shows the parameters of the MCL instruction "setCs()":

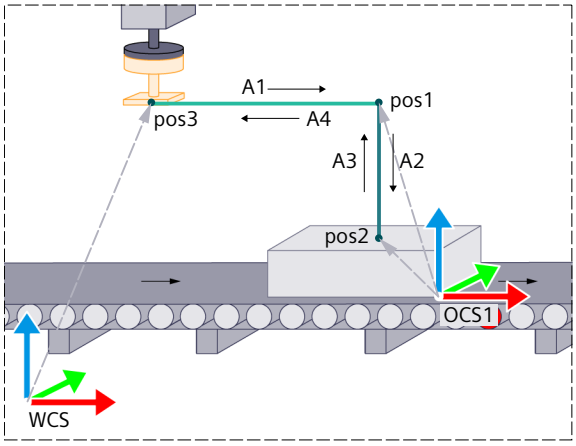
Parameter	Data type	Default value	Description	
cs	DINT	-	Coordinate system	
			0	WCS
			1	OCS1
			2	OCS2
			3	OCS3

Comparable Motion Control instructions

You configure the reference coordinate system for linear and circular path motions and sPTP motions in the motion control instructions in parameter "CoordSystem".

Example: Moving kinematics in different reference coordinate systems

For the labeling of a product, the kinematics is moved from a wait position "pos3" in the WCS to the positions "pos1" and "pos2" in OCS1. For the motion jobs, the OCS1 is set modally as the reference coordinate system with the MCL instruction "setCs()". The reference coordinate system WCS is specified for the motion to the wait position "pos3" in the MCL instruction "linAbs()" (A4).



MCL

```
setCs(1);
// A1 to pos1 in OCS1
linAbs( pos1, trans := 1 );
// A2 to pos2 in OCS1
linAbs( pos2, trans := 0 );
writeVar(insertLabel;
waitEvent(labelInserted);
// A3 to pos1 in OCS1
linAbs( pos1, trans := 1 );
// A4 to pos3 in WCS
linAbs( pos3, cs := 0);
```

8.3 Zones (S7-1500T)

8.3.1 defWsZone() Define workspace zone (S7-1500T)

Description

With the MCL instruction "defWsZone()", you define a workspace zone in relation to the world coordinates system or an object coordinate system. The zones (<TO>.WorkspaceZone[1..10]) defined in the Kinematics technology object are not changed and are available again after a restart of the technology object. The variable "<TO>.StatusWorkspaceZone" in the technology object data block of the connected kinematics contains the workspace zones currently in effect.

The "defWsZone()" job interjects itself in the job sequence on the Kinematics technology object and therefore effective for the following motion jobs.

The MCL instruction offers you the following options:

- Define zone geometry
- Define offset and rotation in the reference coordinate system

You can define up to 10 workspace zones.

You can find more information in the "S7-1500 Kinematics functions" (Page 10) documentation.

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.

Syntax

MCL

```
defWsZone( <ztype>, <nr>, <geometry> [,p1 := <val>] [,p2 := <val>] [,p3 := <val>] [refCs := <val>] [,fr := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "defWsZone()":

Parameters	Data type	Default value	Description	
ztype	DINT	-	Zone type	
			0	Blocked zone
			1	Work zone
			2	Signal zone
nr	DINT	-	Zone number	
			1 ... 10	Zone 1 ... 10
geometry	DINT	-	Zone geometry type	
			0	Cuboid
			1	Sphere
			2	Cylinder
p1	LREAL	0.0	Length x [1]	
			If "geometry" = 1 or 2: Radius [1]	
p2	LREAL	0.0	Length y [1]	
			If "geometry" = 2: Height [1]	
p3	LREAL	0.0	Length z [1]	
refCs	DINT	0	Reference system [1]	
			0	World coordinate system (WCS)
			1	Object coordinate system 1 (OCS1)
			2	Object coordinate system 2 (OCS2)
fr	TO_Struct_Ipr_Frame	-	Offset of the zone zero point in relation to the reference system	

[1] Optional parameter. The absolute value of the specified value is used. If you do not use this parameter in the command call, the default value is used.

Comparable Motion Control instructions

"MC_DefineWorkspaceZone": Define workspace zone

8.3.2 defKinZone() Define kinematics zone (S7-1500T)

Description

With the MCL instruction "defKinZone()", you define a kinematics zone in relation to the tool and flange coordinate system. The zones (<TO>.KinematicsZone[2..10]) defined in the Kinematics technology object are not changed and are available again after a restart of the technology object.

The MCL instruction offers you the following options:

- Define zone geometry
- Define offset and rotation in the reference coordinate system

The "defKinZone()" job is added to the queue of the Interpreter job sequence and therefore affects subsequent motion jobs.

The variable "<TO>.StatusKinematicsZone" in the technology object data block of the connected kinematics contains the kinematics zones currently in effect.

You can define up to 9 kinematics zones. Kinematics zone 1 is the tool center point (TCP) and cannot be changed.

You can find more information in the documentation "S7-1500 Kinematic Functions" (Page 10).

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.

Syntax

MCL

```
defKinZone( <nr>, <geometry> [,p1 := <val>] [,p2 := <val>] [,p3 := <val>] [refCs := <val>]
[,fr := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "defKinZone()":

Parameters	Data type	Default value	Description	
nr	DINT	-	Zone number	
			2 ... 10	Zone 2 ... 10
geometry	DINT	-	Zone geometry type	
			0	Cuboid
			1	Sphere
			2	Cylinder
p1	LREAL	0.0	Length x [1]	
			If "geometry" = 1 or 2: Radius	
p2	LREAL	0.0	Length y [1]	
			If "geometry" = 2: Height	
p3	LREAL	0.0	Length z [1]	
refCs	DINT	0	Reference system	
			0	Flange coordinate system (FCS)
			1	Tool coordinate system (TCS)
fr	TO_Struct_Ipr_Frame	-	Offset of the zone zero point in relation to the reference system	

[1] Optional parameter. The absolute value of the specified value is used. If you do not use this parameter in the command call, the default value is used.

Comparable Motion Control instructions

"MC_DefineKinematicsZone": Define kinematics zone

8.3.3 setWsZoneActive() Activate workspace zone (S7-1500T)

Description

Use the MCL instruction "setWsZoneActive" to activate the zone monitoring for one or more predefined workspace zones.

Use the "mode" parameter to specify whether one or more workspace zones are to be activated. Use the "nr" parameter to define the zone number of the workspace zone that is to be activated with "mode" = 0. If you specify an undefined zone in the "setWsZoneActive" job, the job is rejected and the 1008 technology alarm is triggered. If you do not specify either parameter, the defined workspace zone 1 is activated.

The "setWsZoneActive" job is added to the queue of the Interpreter job sequence and therefore affects subsequent motion jobs.

The "<TO>.StatusWorkspaceZone[1..10].Active" variables in the technology object data block of the connected kinematics contain the current activation status of the zones.

Several defined blocked zones and signal zones can be active at the same time. Only one defined work zone can be active at a time.

You can find more information in the documentation "S7-1500 Kinematic Functions" (Page 10).

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.

Syntax

MCL

```
setWsZoneActive( [mode := <val>] [,nr := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "setWsZoneActive()":

Parameters	Data type	Default value	Description	
mode	DINT	0	Selection of a workspace zone or zone type [1]	
			0	Activate workspace zone defined in the parameter "nr".
			1	Reserved
			2	Activate all defined blocked zones
			3	Activate all defined signal zones
nr	DINT	1	Zone number [1]	
			1 ... 10	Zone 1 ... 10
			If "mode" > 0: The "nr" parameter is ignored	

[1] Optional parameter. The absolute value of the specified value is used. If you do not use this parameter in the command call, the default value is used.

Comparable Motion Control instructions

"MC_SetWorkspaceZoneActive": Activate workspace zone

8.3.4 setWsZoneInactive() Deactivate workspace zone (S7-1500T)

Description

Use the MCL instruction "setWsZoneInactive()" to deactivate zone monitoring for one or more workspace zones.

Use the "mode" parameter to specify whether one or more workspace zones are to be deactivated. Use the "nr" parameter to define the zone number of the workspace zone that is to be deactivated with "mode" = 0. If you do not specify either parameter, the defined workspace zone 1 is deactivated.

The "setWsZoneInactive()" job is added to the queue of the Interpreter job sequence and therefore affects subsequent motion jobs.

The "<TO>.StatusWorkspaceZone[1..10].Active" variables in the technology object data block of the connected kinematics contain the current activation status of the zones.

You can find more information in the documentation "S7-1500 Kinematic Functions" (Page 10).

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.

Syntax

MCL

```
setWsZoneInactive( [mode := <val>] [,nr := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "setWsZoneInactive()":

Para-meters	Data type	Default value	Description	
mode	DINT	0	Selection of a workspace zone or zone type [1]	
			0	Deactivate the zone number defined in the parameter "nr".
			1	Activate all active workspace zones
			2	Deactivate all active blocked zones
			3	Deactivate all active signal zones
			4	Deactivate active work zone
nr	DINT	1	Zone number [1]	
			1 ... 10	Zone 1 ... 10
			If "mode" > 0: The "nr" parameter is ignored	

[1] Optional parameter. The absolute value of the specified value is used. If you do not use this parameter in the command call, the default value is used.

Comparable Motion Control instructions

"MC_SetWorkspaceZoneInactive": Deactivate workspace zone

8.3.5 setKinZoneActive() Activate kinematics zone (S7-1500T)

Description

Use the MCL instruction "setKinZoneActive()" to activate zone monitoring for one or all predefined kinematics zones.

Use the "mode" parameter to specify whether one or all kinematics zones are to be activated. Use the "nr" parameter to define the zone number of the Kinematics zone that is to be activated with "mode" = 0. If you do not specify either parameter, the defined kinematics zone 2 is activated. If an undefined zone is specified in the "setKinZoneActive()" job, the job is rejected and the 1008 technology alarm is triggered.

The "setKinZoneActive()" job is added to the queue of the Interpreter job sequence and therefore affects subsequent motion jobs.

The "<TO>.StatusKinematicsZone[2..10].Active" variables in the technology object data block of the connected kinematics contain the current activation status of the kinematics zones.

You can find more information in the documentation "S7-1500 Kinematic Functions" (Page 10).

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.

Syntax

MCL

```
setKinZoneActive( [mode := <val>] [,nr := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "setKinZoneActive()":

Parameters	Data type	Default value	Description	
mode	DINT	0	Selection of one or more kinematics zones [1]	
			0	Activate kinematics zone defined in the parameter "nr".
			1	All kinematics zones
nr	DINT	2	Zone number [1]	
			2 ... 10	Zone 2 ... 10
			If "mode" > 0: The "nr" parameter is ignored	

[1] Optional parameter. The absolute value of the specified value is used. If you do not use this parameter in the command call, the default value is used.

Comparable Motion Control instructions

"MC_SetKinematicsZoneActive": Activate kinematics zone

8.3.6 setKinZoneInactive() Deactivate kinematics zone (S7-1500T)

Description

Use the MCL instruction "setKinZoneInactive()" to deactivate zone monitoring for one or all active kinematics zones.

Use the "mode" parameter to specify whether one or all active kinematics zones are to be deactivated. Use the "nr" parameter to define the zone number of the Kinematics zone that is to be deactivated with "mode" = 0. If you do not specify either parameter, the defined kinematics zone 2 is deactivated.

The "setKinZoneInactive()" job is added to the queue of the Interpreter job sequence and therefore affects subsequent motion jobs.

The "<TO>.StatusKinematicsZone[2..10].Active" variables in the technology object data block of the connected kinematics contain the current activation status of the kinematics zones.

You can find more information in the documentation "S7-1500 Kinematic Functions" (Page 10).

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.

Syntax

MCL

```
setKinZoneInactive( [mode := <val>] [,nr := <val>] );
```

Parameters

The following table shows the parameters of the MCL instruction "setKinZoneInactive()":

Para-meters	Data type	Default value	Description	
mode	DINT	0	Selection of one or more kinematics zones [1]	
			0	Deactivate kinematics zone defined in the parameter "nr".
			1	All kinematics zones
nr	DINT	2	Zone number [1]	
			2 ... 10	Zone 2 ... 10
			If "mode" > 0: The "nr" parameter is ignored	

[1] Optional parameter. The absolute value of the specified value is used. If you do not use this parameter in the command call, the default value is used.

Comparable Motion Control instructions

"MC_SetKinematicsZoneInactive": Deactivate kinematics zone

8.4 Tools (S7-1500T)

8.4.1 setTool() Change active tool (S7-1500T)

Description

Use the MCL instruction "setTool()" to activate a tool.

A "setTool()" job interrupts the program preparation. Program preparation is restarted after the "setTool()" job has been completed.

You can find more information in the "S7-1500 Kinematics functions" (Page 10) documentation.

Applies to

- Kinematics

Requirements

- The technology objects (Kinematics, Interpreter, Interpreter program) have been configured correctly.
- The kinematics is connected to the Interpreter.
- The axes connected to the kinematics are enabled.
- No single-axis job (e.g. "move()") is active on any of the axes connected to the kinematics.
- The kinematics is at a standstill.
- No kinematics motion is active.

Syntax

MCL

```
setTool( <toolNr> );
```

Parameters

The following table shows the parameters of the MCL instruction "setTool()":

Parameters	Data type	Default value	Description	
toolNr	DINT	-	Number of the tool to be activated.	
			1	Tool 1
			2	Tool 2
			3	Tool 3

Comparable Motion Control instructions

"MC_SetTool": Change active tool

Example: Change active tool

In the following program example, Tool 2 is set as the active tool.

MCL

```
//Set active tool
setTool( 2 );
```

In the technology object data block of the connected kinematics, the number of the active tool is displayed in the following variable:

<TO_Kinematics>.StatusTool.ActiveTool := 2

8.5 Axes (S7-1500T)

8.5.1 Moving() axis with velocity/speed specification (S7-1500T)

Description

With the "move()" MCL instruction, you move an axis at constant velocity/speed.

"move()" is terminated when the velocity setpoint/speed setpoint is reached. The axis then continues to move at the velocity setpoint/speed setpoint.

The MCL instruction offers you the following options:

- Define the velocity setpoint/speed setpoint for the motion
- Define mode for position control
- Define direction of movement
- Define dynamics

The simultaneous start of this instruction with other instructions is allowed. To stop the axis motion at the end of the program, use the instruction "setControlledByInterpreter()" (Page 324).

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Speed axis
- Positioning axis
- Synchronous axis

Requirements

- The Axis technology object has been configured correctly
- The Axis technology object has been enabled
- The Axis technology object is homed

Syntax

MCL

```
move( <axis>, <v> [,pCtrl := <val>] [,a := <val>] [,d := <val>] [,j := <val>] );
```

Parameters

The following table shows the parameters of the "move()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
v	LREAL	-	Velocity setpoint/speed setpoint The sign of the velocity determines the direction of movement.	
pCtrl	DINT	1	Mode for position control[1] This parameter is ignored when a speed axis is used.	
			0	Not in position-controlled mode
			1	Position-controlled mode
a	LREAL	-	Acceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Acceleration" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
d	LREAL	-	Deceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Deceleration" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
j	LREAL	-	Jerk[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Jerk". With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			= 0.0	No jerk limit
			< 0.0	Not permitted

[1] Optional parameter with default value

[2] Modal parameter

Comparable Motion Control instructions

- MC_MoveVelocity: Move axis with velocity/speed specification

Example: Move axis with velocity specification

The following example shows how an axis can be accelerated or decelerated to the specified velocity setpoint.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// "myAxis" is accelerated to the velocity setpoint 75.0
// and modal dynamics values for parameters a, d, j
move( myAxis, 75.0, pCtrl := 0 );
…
// "myAxis" goes to standstill with defined dynamics
move( myAxis, 0.0, d := 3000.0, j := 10000.0 );
// Kinematics axis $A1 is accelerated to the velocity setpoint 10.0
// with negative direction.
// AXIS_OBJECT Literal ($A1) for parameter "axis" is used
move( $A1, -10.0, a := 100.0, d := 1000.0, j := 20000.0 );
```

8.5.2 posAbs() Positioning an axis absolutely (S7-1500T)

Description

With the "posAbs()" MCL instruction, you can move an axis to an absolute position.

The MCL instruction offers you the following options:

- Define absolute target position
- Define direction of movement
- Define dynamics

The simultaneous start of this instruction with other instructions is allowed.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Positioning axis
- Synchronous axis

Requirements

- The Axis technology object has been configured correctly
- The Axis technology object has been enabled
- The Axis technology object is homed

Syntax

MCL

```
posAbs( <axis>, <p> [,dir := <val>] [,v := <val>] [,a := <val>] [,d := <val>]
[,j := <val>] );
```

Parameters

The following table shows the parameters of the "posAbs()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
p	LREAL	-	Absolute target position	
dir	DINT	1	Direction of movement of the axis[1] This parameter is only evaluated when the modulo function is enabled.	
			1	Positive direction
			2	Negative direction
			3	Shortest distance
v	LREAL	-	Velocity[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Velocity" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
a	LREAL	-	Acceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Acceleration" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
d	LREAL	-	Deceleration[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Deceleration" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
j	LREAL	-	Jerk[2]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Jerk". With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			= 0.0	No jerk limit
			< 0.0	Not permitted

[1] Optional parameter with default value

[2] Modal parameter

Comparable Motion Control instructions

- MC_MoveAbsolute: Position axis absolutely

Example: Position axis absolutely

The following example shows how an axis can be moved to the specified absolute target positions.

MCL

```
VAR
   Pos1 : LREAL := 150.0;
END_VAR
…
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// Absolute positioning "myAxis" to position 50.0
posAbs( myAxis, 50.0 );
// Absolute positioning "myAxis" to Pos1 with negative direction and defined dynamics
posAbs( myAxis, Pos1, dir := 2, v := 100.0, a := 2000.0, d := 3000.0, j := 30000.0 );
// Absolute positioning to position 250.0.
// AXIS_OBJECT Literal ($A2) for parameter "axis" is used
posAbs( $A2, 250.0 );
```

8.5.3 posRel() Position axis relatively (S7-1500T)

Description

With the MCL instruction "posRel()", you can move an axis relative to the position that is present at the start of job processing.

The MCL instruction offers you the following options:

- Define relative target position
- Define dynamics

The simultaneous start of this instruction with other instructions is allowed.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Positioning axis
- Synchronous axis

Requirements

- The Axis technology object has been configured correctly
- The Axis technology object has been enabled
- The Axis technology object is homed

Syntax

MCL

```
posRel( <axis>, <p> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] );
```

Parameters

The following table shows the parameters of the "posRel()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
p	LREAL	-	Distance for the positioning process (negative or positive) The sign of the value determines the direction of movement.	
v	LREAL	-	Velocity[1)]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Velocity" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
a	LREAL	-	Acceleration[1)]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Acceleration" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
d	LREAL	-	Deceleration[1)]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Deceleration" With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			≤ 0.0	Not permitted
j	LREAL	-	Jerk[1)]	
			-	The modal value is used. The modal value is initialized with "<TO_Axis>.DynamicDefaults.Jerk". With setAxisDyn() (Page 299), you can set another value modally.
			> 0.0	The specified value is used.
			= 0.0	No jerk limit
			< 0.0	Not permitted

1) Modal parameter

Comparable Motion Control instructions

- MC_MoveRelative: Position axis relatively

Example: Position axis relatively

The following example shows how to move an axis to target positions that are specified relative to the current position.

MCL

```
VAR
   Pos1 : LREAL := 150.0;
END_VAR
…
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// "myAxis" is positioned relatively to the current position,
// using the relative target position
posRel( myAxis, 50.0 );
// "myAxis" is positioned relatively to the current position,
// using the relative target position with defined dynamics
posRel( myAxis, Pos1, v := 100.0, a := 2000.0, d := 3000.0, j := 30000.0 );
// "myAxis" is positioned relatively to the current position,
// using the relative target position with negative direction
// AXIS_OBJECT Literal ($A2) for parameter "axis" is used
posRel( $A2, -250.0 );
```

8.5.4 setAxisDyn() Set dynamic defaults for single-axis motions modally (S7-1500T)

Description

With the MCL instruction "setAxisDyn()", you can set the modal values for the dynamic default of the single-axis motions. If you do not specify any other dynamic values for an MCL motion job, these modal dynamic values are used for the single-axis motions.

You can specify the values for one or more modal dynamic parameters. Any dynamic value not specified in the instruction remains unchanged.

Specify a value for at least one of the dynamic parameters.

You can find more information on the maximum limits in the "Modal parameters (Page 179)" section.

Applies to

- Positioning axis
- Synchronous axis

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- posAbs()
- posRel()
- move()

Syntax

MCL

```
setAxisDyn( <axis> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>]
[,rel := <val>] );
```

Parameters

The following table shows the parameters of the "setAxisDyn()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
v	LREAL	-	Modal value of the velocity of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicDefaults.Velocity). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default setting of the velocity
			0.0 < v ≤ 100.0	With rel := TRUE: Relative default setting as a percentage of the last valid maximum velocity value
			≤ 0.0	Not permitted
a	LREAL	-	Modal value of the acceleration of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicDefaults.Acceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the acceleration

[1] Optional parameter with default value

Parameter	Data type	Default value	Description	
a	LREAL	-	0.0 < a ≤ 100.0	With rel := TRUE: Relative default setting as a percentage of the last valid maximum acceleration value
			≤ 0.0	Not permitted
d	LREAL	-	Modal value of the deceleration of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicDefaults.Deceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the deceleration
			0.0 < d ≤ 100.0	With rel := TRUE: Relative default setting as a percentage of the last valid maximum deceleration value
			≤ 0.0	Not permitted
j	LREAL	-	Modal value of the jerk of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicDefaults.Jerk). If the value is not set in the program code, the initialized value is used.
			> 0.0	With rel := FALSE: Absolute default of the jerk
			0.0 < j ≤ 100.0	With rel := TRUE: Relative default setting as a percentage of the last valid maximum jerk
			= 0.0	No jerk limit
			< 0.0	Not permitted
rel	BOOL	FALSE	Type of specification of the modal value[1)]	
			FALSE	Absolute specification
			TRUE	Percentage specification

1) Optional parameter with default value

Example

The following example shows how to set the modal values of the dynamic parameters of a single axis.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// Set the modal values of the dynamic parameters of "myAxis"
setAxisDyn( myAxis, v := 120.0, a := 1000.0, d := 1000.0, j := 0.0 );
// Set the modal value of axis velocity to absolute 10.0
setAxisDyn( $A1, v := 10.0 );
// Set modal value of axis acceleration to 50% of maximum value
setAxisDyn( $A2, a := 50.0, rel := TRUE );
```

8.5.5 setAxisDynMax() Set dynamic limits for single-axis motions modally (S7-1500T)

Description

With the MCL instruction "setAxisDynMax()", you can set the maximum limits for the modal dynamic parameters of single-axis motions.

You can set the maximum dynamic values for one or more modal parameters. Maximum dynamic values can only be specified in absolute values. Any dynamic value not specified in the instruction remains unchanged.

Specify a value for at least one of the dynamic parameters.

You can find more information on dynamic defaults in the "Modal parameters (Page 179)" section.

Applies to

- Positioning axis
- Synchronous axis

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- posAbs()
- posRel()
- move()

Syntax

MCL

```
setAxisDynMax( <axis> [,v := <val>] [,a := <val>] [,d := <val>] [,j := <val>] );
```

Parameters

The following table shows the parameters of the "setAxisDynMax()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
v	LREAL	-	Modal value of the maximum velocity of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicLimits.MaxVelocity). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of velocity limits
			≤ 0.0	Not permitted
a	LREAL	-	Modal value of the maximum acceleration of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicDefaults.MaxAcceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the limits of acceleration
			≤ 0.0	Not permitted
d	LREAL	-	Modal value of the maximum deceleration of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicDefaults.MaxDeceleration). If the value is not set in the program code, the initialized value is used.
			> 0.0	Absolute default of the deceleration limits
			≤ 0.0	Not permitted
j	LREAL	-	Modal value of the maximum jerk of the single-axis motion	
			-	The modal value remains unchanged. The last valid value set in the program code is used. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Dynamics" (<TO_Axis>.DynamicLimits.MaxJerk). If the value is not set in the program code, the initialized value is used.

Para-meter	Data type	Default value	Description	
j	LREAL	-	> 0.0	Absolute default of the jerk limits
			= 0.0	No jerk limit
			< 0.0	Not permitted

Example

The following example shows how to set the maximum limits of the dynamic parameters of a single axis.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// Set the maximum values of the dynamic parameters of "myAxis"
setAxisDynMax( myAxis, v := 500.0, a := 10000.0, d := 10000.0, j := 70000.0 );
// Set the maximum value of axis velocity to absolute 10.0
setAxisDynMax( $A1, v := 10.0 );
```

8.5.6 powerOn() Enable axis (S7-1500T)

Description

With the MCL instruction "powerOn()", you enable one axis or all connected kinematics axes and can change the position control mode.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Speed axis
- Positioning axis
- Synchronous axis
- All connected kinematics axes

Requirements

- The Axis technology object has been configured correctly
- "Drive ready" is set in the drive ("<TO_Axis>.StatusDrive.InOperation" = TRUE)
- Cyclic bus communication is established between controller and encoder ("<TO_Axis>.StatusSensor[1..4].CommunicationOK" = TRUE)
- Cyclic bus communication is established between controller and drive ("<TO_Axis>.StatusDrive.CommunicationOK" = TRUE)
- The status of the active encoder is valid ("<TO_Axis>.StatusSensor[1..4].State" = 2)
- The optional data adaptation is completed ("<TO>.StatusDrive.AdaptionState" = 2 and "<TO>.StatusSensor[1..4].AdaptionState" = 2)

Override response

- "powerOn()" cannot be canceled by any other Motion Control job.
- "powerOn()" releases a technology object and does not cancel any other Motion Control instructions.

Syntax

MCL

```
powerOn( [axis := <val>] [,mode := <val>] );
```

Parameters

The following table shows the parameters of the "powerOn()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	NULL	Technology object[1)] If the optional "axis" parameter is not specified or has the value NULL, the instruction refers to all assigned kinematics axes.	
mode	DINT	1	Mode for position control[1)] This parameter is ignored when a speed axis is used.	
			0	Not in position-controlled mode
			1	Position-controlled mode

1) Optional parameter with default value

Comparable Motion Control instructions

- MC_Power: Enable technology object (if "Enable" = TRUE)

Example

The following example shows how to enable a single axis and all assigned kinematics axes.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in <TO_InterpreterMapping>
// enable mapped "myAxis" and change the control mode to non-position controlled mode
powerOn( axis := myAxis, mode := 0 );
…
// enable axis $A3. AXIS_OBJECT Literal ($A3) for parameter "axis" is used
powerOn( axis := $A3 );
…
// enable all assigned kinematics axes
powerOn();
```

8.5.7 powerOff() Disable axis (S7-1500T)

Description

With the MCL instruction "powerOff()", you can disable one axis or all connected kinematics axes according to the selected stop mode.

"powerOff()" generates an implicit preHalt in the program execution.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Speed axis
- Positioning axis
- Synchronous axis
- All connected kinematics axes

Requirements

- The Axis technology object has been configured correctly
- "Drive ready" is set in the drive ("<TO_Axis>.StatusDrive.InOperation" = TRUE)
- Cyclic bus communication is established between controller and encoder ("<TO_Axis>.StatusSensor[1..4].CommunicationOK" = TRUE)
- Cyclic bus communication is established between controller and drive ("<TO_Axis>.StatusDrive.CommunicationOK" = TRUE)
- The status of the active encoder is valid ("<TO_Axis>.StatusSensor[1..4].State" = 2)
- The optional data adaptation is completed ("<TO>.StatusDrive.AdaptionState" = 2 and "<TO>.StatusSensor[1..4].AdaptionState" = 2)

Override response

- "powerOff()" cannot be canceled by any other Motion Control job.
- "powerOff()" cancels all motion jobs at the associated technology object according to the selected "Stop mode". This process cannot be canceled by the user.

Syntax

MCL

```
powerOff( [axis := <val>] [,mode := <val>] );
```

Parameters

The following table shows the parameters of the "powerOff()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	NULL	Technology object[1)] If the optional "axis" parameter is not specified or has the value NULL, the instruction refers to all assigned kinematics axes.	
mode	DINT	0	Stop mode[1)]	
			0	Emergency stop When the technology object is disabled, the axis is braked to a standstill without jerk limit, using the emergency stop deceleration configured in "Technology object > Configuration > Extended parameters > Emergency stop". The drive is then switched off and the technology object is disabled. (<TO_Axis>.DynamicDefaults.EmergencyDeceleration)
			1	Immediate stop When a technology object is disabled, the setpoint zero is output. The axis is decelerated to a standstill according to the configuration in the drive. The drive is then switched off and the technology object is disabled.
			2	Stop with maximum dynamic values When the technology object is disabled, the axis is braked to a standstill using the maximum deceleration configured in "Technology object > Configuration > Extended parameters > Dynamic limits". The configured maximum jerk is hereby taken into account. The drive is then switched off and the technology object is disabled. (<TO_Axis>.DynamicLimits.MaxDeceleration; <TO_Axis>.DynamicLimits.MaxJerk)

[1)] Optional parameter with default value

Para-meter	Data type	Default value	Description	
mode	DINT	0	3	Stop with specified dynamic response When the technology object is disabled, the drive is de-energized (pulse inhibit) and goes into the closing lockout state. The drive then coasts to a stop. If you are using a drive with an analog setpoint interface, the enable output is disabled and the analog output signal is set to 0.0.

1) Optional parameter with default value

Comparable Motion Control instructions

- MC_Power: Enable technology object (if "Enable" = FALSE)

Example

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in <TO_InterpreterMapping>
// disabling mapped axis "myAxis" with max. dynamics values
powerOff( axis := myAxis, mode := 2 );
…
// disabling axis $A3 with emergency stop.
// AXIS_OBJECT Literal ($A3) for parameter "axis" is used
powerOff( axis := $A3 );
…
// disable all assigned kinematics axes
powerOff();
```

8.5.8 Home() axis, set reference point (S7-1500T)

Description

With the MCL instruction "home()", you can establish the relationship between the position of the axis and the mechanical position. The position value of the axis is assigned to a homing mark. This homing mark represents a known mechanical position.

With active homing, the default values under "Extended parameters > Dynamic default values" are used for the dynamic values acceleration, deceleration, and jerk.

"home()" is terminated when the reference position is set or reached, depending on the selected mode.

The MCL instruction offers you the following options:

- Define home position
- Set homing mode

The simultaneous start of this instruction with other instructions is allowed.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Positioning axis
- Synchronous axis

Requirements

- The Axis technology object has been configured correctly
- The Axis technology object is enabled (for "mode" = 3, 5)
- The actual encoder values are valid ("<TO_Axis>.StatusSensor[1..4].State" = 2 for "mode" = 6, 7, 11, 12, 13)
- The axis is in position-controlled mode (for "mode" = 0, 1, 6, 7)

Syntax

MCL

```
home( <axis> [,mode := <val>] [,sensor := <val>] [,p := <val>] );
```

Parameters

The following table shows the parameters of the "home()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
mode	DINT	0	Homing mode[1)]	
			0	Direct homing (absolute) The current position of the technology object is set to the value of parameter "p". **Note:** In the case of an axis with several encoders, the offset of the position at the sensors of all encoders is also applied with a position correction with the parameter "mode" = 0. This prevents the sensors from diverging.
			1	Direct homing (relative) The current position of the technology object is shifted by the value of parameter "p". **Note:** In the case of an axis with several encoders, the offset of the position at the sensors of all encoders is also applied with a position correction with the parameter "mode" = 1. This prevents the sensors from diverging.

1) Optional parameter with default value

Parameter	Data type	Default value	Description	
mode	DINT	0	3	Active homing The technology object performs a homing movement according to the configuration. After completion of the motion, the axis is positioned at the value of the "p" parameter.
			5	Active homing (home position, parameter "p" ineffective) The technology object performs a homing movement according to the configuration. After completion of the motion, the axis is positioned at the home position configured under "Technology object > Configuration > Extended parameters > Homing > Active homing" (<TO_Axis>.Homing.HomePosition).
			6	Absolute encoder adjustment (relative) The current position is shifted by the value of parameter "p". The calculated absolute value offset is stored retentively in the CPU. (<TO_Axis>.StatusSensor[1..4].AbsEncoderOffset)
			7	Absolute encoder adjustment (absolute) The current position is set to the value of parameter "p". The calculated absolute value offset is stored retentively in the CPU. (<TO_Axis>.StatusSensor[1..4].AbsEncoderOffset)
			11	Setting of position setpoint (absolute) The position setpoint of the technology object is set to the value of the "p" parameter. The following error remains.
			12	Shift of position setpoint (relative) The position setpoint of the technology object is shifted by the value of the "p" parameter. The following error remains.
			13	Incremental encoder adjustment The current position is set to the value of parameter "p".
sensor	DINT	0	Selection of the absolute encoder ("mode" = 6, 7) or incremental encoder ("mode" = 13) to be calibrated[1)]	
			0	Operative encoder
			1..4	Encoder 1..4
p	LREAL	0.0	Home position[1)] The specified value is used according to the selected "mode".	

1) Optional parameter with default value

Comparable Motion Control instructions

- MC_Home: Home technology object, set home position

Example: Home axis

The following example shows how to perform homing with the configured settings.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// Homing "myAxis" with default values
home( myAxis );
…
// Parameter p is not relevant for this MCL-command
// Position is taken from "myAxis.Homing.HomePosition"
home( myAxis, mode := 5 );
…
// The current position of kinematics axis $A3 is set to the value 100.0
home( $A3, mode := 0, p := 100.0 );
```

8.5.9 torqueLimitOn() Activate force/torque limiting/fixed stop detection (S7-1500T)

Description

With the MCL instruction "torqueLimitOn()", you can activate and configure the force/torque limiting or fixed stop detection. Together with a position-controlled motion job, a "Travel to fixed stop" can be realized with the fixed stop detection. In the axis configuration, you can configure whether the force/torque limiting is to relate to the drive side or the load side.

"torqueLimitOn()" can be activated before and during an active motion job.

"torqueLimitOn()" is terminated when the force/torque limit or the fixed stop detection is activated. If a fixed stop is detected, the current drive command is canceled. The program pointer is then switched to the next line.

"torqueLimitOn()" cannot be cancelled by any other MCL or MC command.

The MCL instruction offers you the following options:

- Define the value of the force/torque limiting.
- Define force/torque limit mode

The simultaneous start of this instruction with other instructions is not allowed.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Force/torque limiting applies to

- Speed axis
- Positioning axis
- Synchronous axis

Requirements for force/torque limiting

- The axis and the reference torque of the drive have been configured correctly.
- The axis has been released
- The drive must support force/torque reduction. Only PROFIdrive drives with SIEMENS telegram 10x support force/torque limiting.
- Interconnection in the SINAMICS drive:
  - P1522 to a fixed value of 100%
  - P1523 to a fixed value of -100% (e.g. through interconnection to fixed value parameter P2902[i])
  - P1544 Torque/force reduction evaluation during travel to fixed stop to 100% (default)
  - P2194 Threshold value for the parameter "InLimitation" of <100% (default value 90%)

Fixed stop detection applies to

- Positioning axis
- Synchronous axis

Requirements for fixed stop detection

- Fixed stop detection can only be applied to position-controlled axes. For fixed stop detection, the axis must be enabled as position-controlled. Motion jobs must be executed as position-controlled.
- The axis has been configured correctly.
- When a drive and telegram that support force/torque limiting are used, the reference torque of the drive must be correctly configured for the axis.
- No errors that prevent enabling are pending at the axis (the axis must be enabled).

Syntax

MCL

```
torqueLimitOn( <axis>, [,limit := <val>] [,mode := <val>] );
```

Parameters

The following table shows the parameters of the "torqueLimitOn()" instruction:

Parameter	Data type	Default value	Description	
axis	AXIS_OBJECT	-	Technology object	
limit	LREAL	-1.0	Value of force/torque limiting (in the configured unit)[1] If the drive and telegram do not support force/torque limiting, the specified value is irrelevant.	
			≥ 0	The value specified at the parameter is used.
			< 0	The configured value in the "Torque limit" configuration window is used. Variable limit value torque: <TO_Axis>.TorqueLimiting.LimitDefaults.Torque Variable limit value force: <TO_Axis>.TorqueLimiting.LimitDefaults.Force
mode	DINT	0	Force/torque limiting mode[1] If the drive and telegram do not support force/torque limiting, the specified value is irrelevant.	
			0	Force/torque limiting
			1	Fixed stop detection Only relevant for position-controlled axes.

[1] Optional parameter with default value

Comparable Motion Control instructions

- MC_TorqueLimiting: Activate/deactivate force/torque limit/fixed stop detection (if "Enable" = TRUE)

Example: Activate fixed stop detection

The following example shows how to activate fixed stop detection.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// The axis is enabled as position-controlled
// The torque limit with fixed stop detection of the "myAxis" is activated
torqueLimitOn( myAxis, limit := -1.0, mode := 1 );
// "myAxis" is moved to fixed stop. When the following error limit is reached,
// the positioning is canceled
posAbs( myAxis, 100.0 );
```

8.5.10 torqueLimitOff() Deactivate force/torque limiting/fixed stop detection (S7-1500T)

Description

With the MCL instruction "torqueLimitOff()", you can disable force/torque limiting or fixed stop detection.

"torqueLimitOff()" can be activated before and during an active motion job.

"torqueLimitOff()" is terminated when the force/torque limiting or the fixed stop detection is deactivated. If a fixed stop is detected, the current drive command is canceled. The program pointer is then switched to the next line.

"torqueLimitOff()" cannot be canceled by any other MCL or MC command.

The simultaneous start of this instruction with other instructions is not allowed.

You can find more information in the "S7-1500/S7-1500T Axis functions (Page 10)" documentation.

Applies to

- Speed axis
- Positioning axis
- Synchronous axis

Syntax

MCL

```
torqueLimitOff( <axis> );
```

Parameters

The following table shows the parameters of the instruction "torqueLimitOff()":

Parameter	Data type	Default value	Description
axis	AXIS_OBJECT	-	Technology object

Comparable Motion Control instructions

- MC_TorqueLimiting: Activate/deactivate force/torque limit/fixed stop detection (if "Enable" = FALSE)

Example: Deactivating the torque limiting

The following example shows how to disable torque limiting for an axis.

MCL

```
// The mapping of <TO_Axis> to "myAxis" tag is done in the <TO_InterpreterMapping>
// The torque limit of the "myAxis" is deactivated
torqueLimitOff( myAxis );
```

8.6 Other instructions (S7-1500T)

8.6.1 writeVar() Write mapped PLC tag or technology object data block tag (S7-1500T)

Description

The "writeVar" instruction in the Interpreter program writes the value of an expression to a variable in the context of program execution. The expression in the <expression> argument is evaluated in the program preparation and used for value assignment in the program execution.

Note that the assigned operation is executed with the ":=" operator in the program preparation.

Applies to

- Technology object data block tag of the Interpreter or assigned objects
- Mapped PLC variable

Syntax

MCL

```
writeVar( <var>, <expression> );
```

Parameter	Declaration	Data type	Default value	Description
var	INPUT	-	-	Tag to be written: • Technology object data block tag of the Interpreter or assigned objects • Mapped PLC variable
expression	INPUT	-	-	The value of an expression written to a variable.

Rules

- Writing is limited to a single data access.
- Using the "writeVar" instruction in the SYNC block between two instructions of a path motion with or without blending is not allowed (see example 2 and section "SYNC instruction (Page 133)").

Example 1

In this example, the clipboard variable "$IPR.clipboard.cbDint[1]" is written to the program execution with the instruction "writeVar".

MCL

```
...

linAbs( pos1 );


// execute in the main task; kinematics has reached position pos1
writeVar( $IPR.clipboard.cbDint[1], 2 );

...
```

Example 2

In this example, "plcVar" has the value "1" before program execution (line 1). This value is cached and used during program execution for writing the tag "CbDint[1]" in the "writeVar" commands (line 2 and line 4) before the program execution assigns the value "2" to the tag "plcVar" (line 3). The resulting value of "CbDint[1]" is "1".

MCL

```
...

plcVar := 1; // line 1, plcVar – mapped PLC variable

writeVar( $IPR.Clipboard.CbDint[1], plcVar ); // 2

writeVar( plcVar, 2 ); // 3

writeVar( $IPR.Clipboard.CbDint[1], plcVar); // 4

...
```

Example 3

In this example, position-triggered synchronous actions are programmed during a path motion with blending. The programming of the "writeVar" instruction is invalid. An error message occurs in the Interpreter technology object and the Interpreter program closes.

MCL

```
SYNC // line 1
 linAbs( myPos, trans := 0 ); // 2 path movement
 writeVar( $IPR.clipboard.cbDint[1], 1 ); // 3 invalid command between SYNC and ON_POS
 linAbs( myPos2, trans := 0 ); // 4
 ON_POS sType := 2, p := 25.0, t := T#-1s DO // 5 referred to line 2
   writeVar( $IPR.clipboard.cbDint[1], 2 ); // 6 execute at position
 ON_POS sType := 2, p := 50.0, t := T#1s DO // 7 referred to line 2
   writeVar( $IPR.clipboard.cbDint[1], 4 ); // 8 execute at the end
END_SYNC;
```

8.6.2 waitTime() Interrupt program execution for a defined period (S7-1500T)

Description

The "waitTime" instruction in the Interpreter program enables the execution of the Interpreter program to be interrupted for the time period specified in the instruction parameter.

Applies to

- Program organization unit "Main program" of the Interpreter program
- Program organization unit "Function" of the Interpreter program

Syntax

MCL

```
waitTime( <time> );
```

Parameter	Declaration	Data type	Default value	Description
time	INPUT	TIME	-	Wait time

Rule

- Negative time intervals result in a runtime error.
- Maximum time resolution is 1 cycle of the cycle time of the "MC_Interpolator" (T_{IPO})

Example

In the following example, the Interpreter program interrupts for 3 seconds:

MCL

```
// interruption of command interpretation for 3 s
waitTime( T#3000ms );
```

8.6.3 waitEvent() Interrupt program execution until a specific event (S7-1500T)

Description

The "waitEvent" instruction stops program execution until either a specific external event occurs or an optionally programmable time period has elapsed. This instruction in the Interpreter program can be used for the following:

- Executing motion after an external signal is triggered.
- Adding motion jobs to an active motion, depending on an external event.

If the event condition is TRUE, the program execution is continued.

Syntax

MCL

```
waitEvent( <event> [,mode := <val>] [,timeout := <val>] [,timeoutState => <val>] );
```

Parameter	Declaration	Data type	Default value	Description	
event	INPUT	BOOL	-	Result condition as: • Tag of the BOOL data type The following are possible: – Technology object data block tags of the Interpreter – Tags of the technology object data block of an assigned object – Mapped PLC tags • Logical expression The logical expressions may include: – Constants in the Interpreter program – Tags of the technology object data block of the Interpreter – Tags of the technology object data block of an assigned object – Mapped PLC tags – Calls of standard functions, e.g. Abs(), Sin()	
				Not permitted: • User functions • Tags declared in the Interpreter program	
mode	INPUT	DINT	0	Processing mode[1)]	
				0	Event only checked in the program execution.

1) Optional

Parameter	Declaration	Data type	Default value	Description	
mode	INPUT	DINT	0	1	Additionally to "mode" = "0": checked of result in the program preparation and in the program execution
timeout	INPUT	TIME	T#-1ms	Maximum wait time[1)] Wait time from the program execution of the instruction, after which further preparation and execution jobs are continued, regardless of the event condition.	
				< T#0ms	No consideration of the wait time
				= T#0ms	Immediate program execution
				> T#0ms	Maximum wait time
timeoutState	OUTPUT	BOOL	FALSE	Output tag[1)]	
				FALSE	Wait time not exceeded
				TRUE	Wait time exceeded

1) Optional

Rules

- The "timeout" parameter takes effect if the event condition is not met in preparation. Further program execution is stopped until the event condition is fulfilled ("event" parameter) or until the wait time specified in the "timeout" parameter has elapsed. The wait time starts with the end of the previous job in the program execution. After the specified time, preparation and execution continue, regardless of the event condition.
- If the "timeout" is set to T#0ms, there is no waiting time for the event condition to be fulfilled, and further program execution continues immediately.
- If a negative "timeout" parameter is set, the program execution waits for the event condition to be fulfilled without any time limit.
- The "timeoutState" output parameter can be used to check whether a "timeout" has occurred. If this is the case, the corresponding output tag returns TRUE.
- If the event condition is fulfilled while "waitEvent" with parameter "mode" = 1 is in the program preparation phase, the program preparation continues immediately. If the event condition is not fulfilled during the preparation, then "waitEvent" and thus the cyclic check of the event in the program execution is active.
- If "waitEvent" with "mode" = 1 follows a motion job with blending, the following behavior is defined:
  - If the event condition is fulfilled in time before the preceding motion job is finished, the programmed blending motion is executed according to the program sequence.
  - If the event condition is fulfilled too late, an exact stop is executed.
- If "waitEvent" with "mode" = 0 follows a motion job with blending, an exact stop is executed.

Example 1

In this example, "waitEvent" (parameter "mode" = 0 as default value) is programmed after a path job with blending. Blending of the instruction is not possible, because the "waitEvent" is being executed in the program execution. After an exact stop at pos1, the execution of the further instructions pauses until the interface tag "$IPR.Clipboard.cbBool[1]" = TRUE.

MCL

```
linAbs( pos1, trans := 1 );
waitEvent( $IPR.Clipboard.cbBool[1] );
linAbs( pos2, trans := 0 );
```

Example 2

In this example, "waitEvent" with programmed "mode" = 1 occurs after an instruction with blending. If the result ("$IPR.Clipboard.cbBool[1]" tag returns TRUE) is fulfilled in time during the current motion, the path motion to pos3 with blending can be executed according to the programming, otherwise an exact stop is executed.

If the advance condition does not occur during the current motion, an exact stop is executed at pos2. In this case, the system waits at pos2 until the system tag of the Interpreter "$IPR.Clipboard.cbBool[1]" = TRUE or until the wait time has elapsed.

If the program execution is blocked by the unfulfilled step-enabling condition for more than 3 seconds ("timeout" = T#3s), the program decoding and execution continues irrespective of the step-enabling condition. In this case, the motion from position 2 to position 3 is executed without blending.

MCL

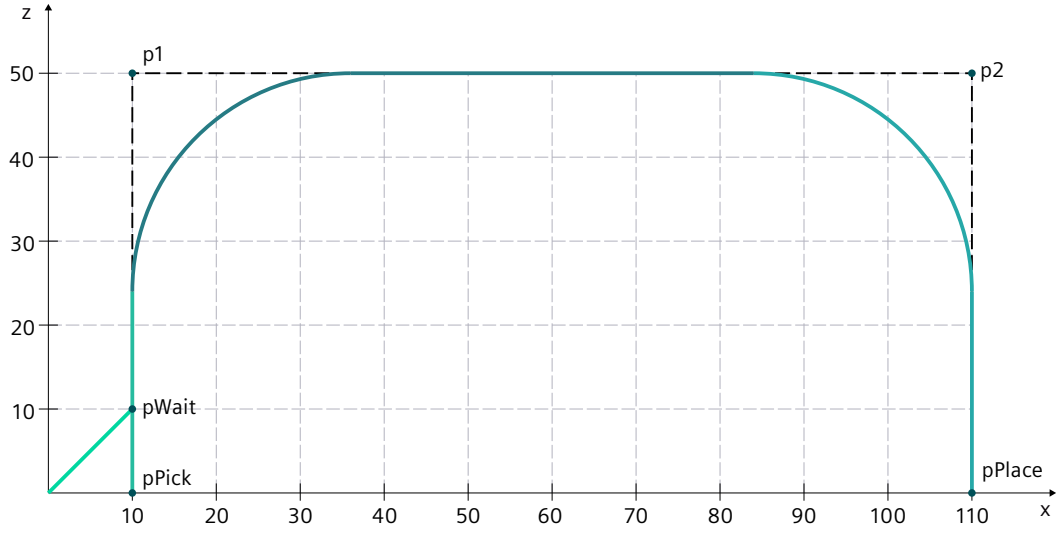
```
setBlend( 2 );
linAbs( pos1, trans := 1 );
linAbs( pos2, trans := 1 );
waitEvent( $IPR.Clipboard.cbBool[1], mode := 1,
           timeout := T#3s, timeoutState => isTimeout );
linAbs( pos3, trans := 0 );
```

Example 3

In this example, the kinematics move to a waiting position "pWait". Through the writing of the mapped PLC tag "plcOpenGripper", the gripper open.

"waitEvent" then waits for the feedback "plcIsGripperOpened" of the gripper. After the opening of the gripper, the kinematics moves to a pick position "pPick" and the gripper closes through the writing of the mapped PLC tag "plcCloseGripper".

"waitEvent" then waits for the feedback "plcIsGripperClosed" of the gripper. After the closing of the gripper, the kinematics moves to the place position "pPlace" and the gripper opens again.



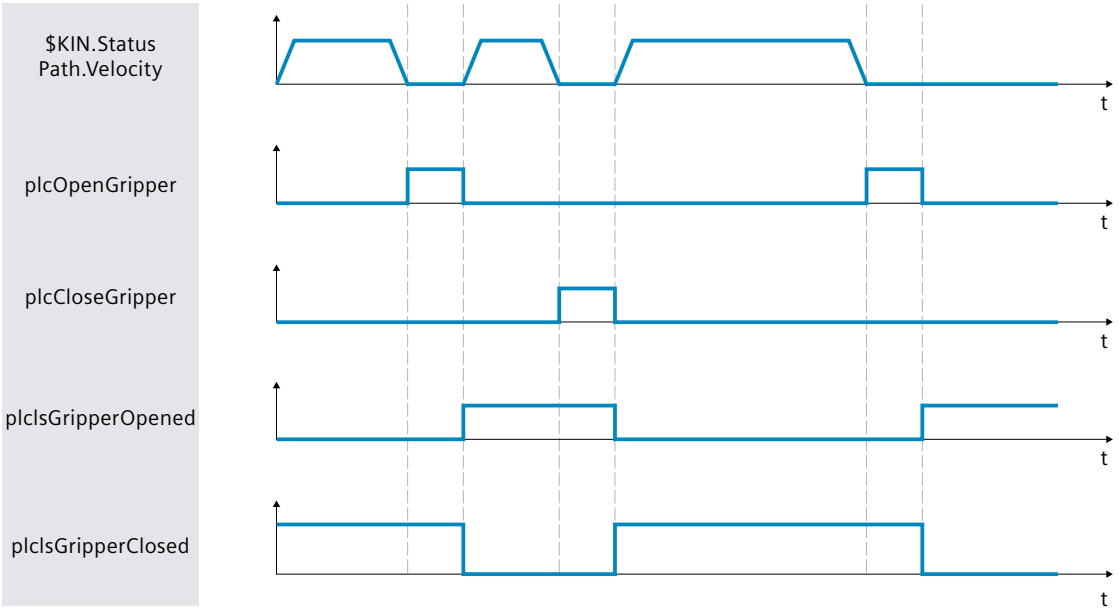
MCL

```
VAR

 pWait : TO_Struct_Ipr_Position := ( x := 10.0, y := 0.0, z := 10.0 );

 pPick : TO_Struct_Ipr_Position := ( x := 10.0, y := 0.0, z := 0.0 );
 pPlace : TO_Struct_Ipr_Position := ( x := 110.0, y := 0.0, z := 0.0 );
 p1 : TO_Struct_Ipr_Position := ( x := 10.0, y := 0.0, z := 50.0 );
 p2 : TO_Struct_Ipr_Position := ( x := 110.0, y := 0.0, z := 50.0 );

END_VAR
…
 setBlend( 2 ); // set modal values
 setBlendDist( 30.0 );
 setTrans( 0 );
 setCs( 0 ); // set wcs modally
…
 linAbs( pWait );
 writeVar( plcOpenGripper, TRUE );
 waitEvent( plcIsGripperOpened ); // wait for gripper opened
 linAbs( pPick );
 writeVar( plcOpenGripper, FALSE );
 writeVar( plcCloseGripper, TRUE );
 waitEvent( plcIsGripperClosed ); // wait for gripper closed
 linAbs( p1, trans := 1, blend := 2 );
 linAbs( p2, trans := 1, blend := 2 );
 linAbs( pPlace );
 writeVar( plcOpenGripper, TRUE );
 waitEvent( plcIsGripperOpened ); // wait for gripper opened
…
```



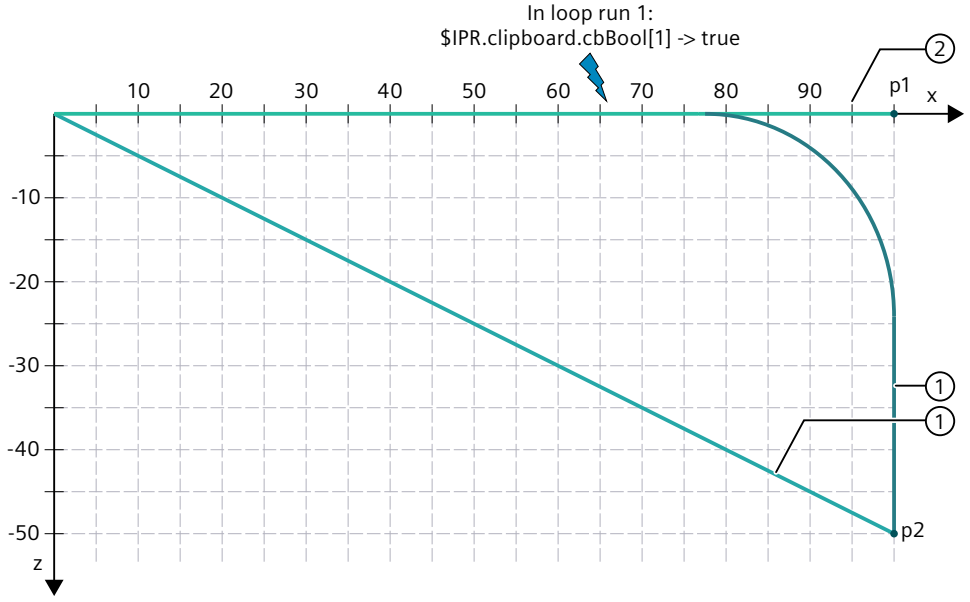
Example 4

In this example, an external event occurs in the first loop pass (i = 1) of the FOR instruction during the execution of the linear motion of the kinematics. The clipboard tag "$IPR.clipboard.cbBool[1]" detects the external event.

The event (setting of the "$IPR.Clipboard.cbBool[1]" tag to TRUE) occurs during the current motion ("linAbs()" to "p1") in order to execute the next instruction and to blend into the linear motion ("linAbs()" to "p2"). Then a linear motion is executed ("linAbs()" to "(x = 0, y = 0, z = 0)").

In the second loop run of the FOR instruction (i = 2), if the external event did not occur during the current motion ("linAbs()" to "p1"), the motion is nevertheless executed until "p1" and stopped there.

"waitEvent" then waits until the programmed timeout time elapses ("timeout" = T#1s) for the fulfilled event condition in the program execution. Since the event does not occur within 1 second, the output tag "isTimeout" is set to TRUE. The subsequent instruction "preHalt" is required to synchronize execution and preparation.



① In loop run 1

② In loop run 2

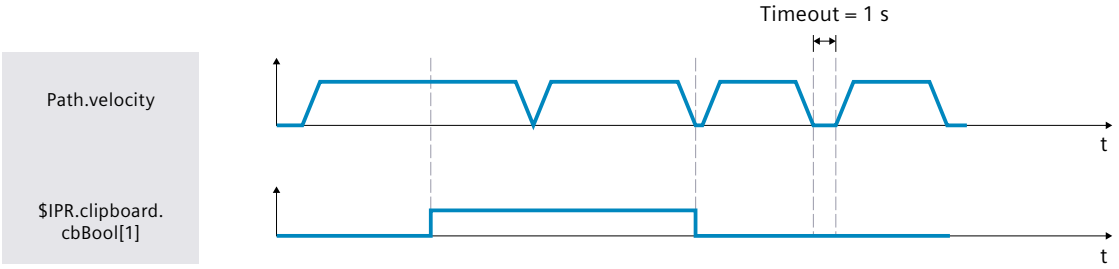
MCL

```
 VAR
   i : DINT := 0;
   isTimeout : Bool := FALSE;
   p1 : TO_Struct_Ipr_Position;
   p2 : TO_Struct_Ipr_Position;
 END_VAR
…
 p1 := ( x := 100.0, y := 0.0, z := 0.0, A := 0.0 );
 p2 := ( x := 100.0, y := 0.0, z := -50.0, A := 0.0 );
…
 setBlendDist( 30.0 );
 setBlend( 2 );
 setTrans( 0 );
 setCs( 0 );
…
 FOR i := 1 TO 2 DO
   preHalt(); // synchronize execution and preparation

   isTimeout := FALSE; // reset timeout tag
   writeVar($IPR.clipboard.cbBool[1], FALSE ); // reset event before start of movement


   linAbs( p1, blend := 2, trans := 1 ); // start movement
   waitEvent( $IPR.clipboard.cbBool[1], mode := 1, // wait for event, check
    timeout := T#1s, timeoutState => isTimeout); // starts in preparation

   IF isTimeout THEN
     linAbs( (x := 0.0, y := 0.0, z := 0.0) );
   ELSE
     linAbs( p2 );
     linAbs( (x := 0.0, y := 0.0, z := 0.0) );
   END_IF;
 END_FOR;
```



Example 5 with logical expression

In this example, the kinematics axis is enabled outside the Interpreter program.
The axis enable is queried in the Interpreter program with a waitEvent instruction. Bit 0 of the status word is evaluated with a logical expression.

If the axis has not been enabled for program preparation of the waitEvent job, the program execution is continued after the maximum wait time of 1 s. The subsequent motion job is not executed because of the missing axis enable.

MCL

```
 VAR
   waitEventstate: Bool;
 END_VAR
 VAR CONSTANT
   axEnabled: DWORD := 16#01;
END_VAR

 WaitEvent( ($A1.Statusword AND axEnabled) = axEnabled,
T#1000ms, timeoutState => waitEventState);
 linRel( (x := 100.0), trans := 1);
 linRel( (x := -100.0) )
…
```

8.6.4 setControlledByInterpreter() Set "ControlledByInterpreter" bit for a technology object (S7-1500T)

Description

The "setControlledByInterpreter" instruction enables setting or resetting of the "ControlledByInterpreter" Status-Bit (<TO>.StatusInterpreterMotion.StatusWord.X0) of technology objects.

The "ControlledByInterpreter"-Status-Bit indicates whether the corresponding technology object (Axis or Kinematics) is controlled by the Interpreter technology object (via MCL instructions). The interaction of MCL instructions from the Interpreter program with PLCopen instructions depends on the ControlledByInterpreter status at the axes/kinematics.

The "ControlledByInterpreter" status = TRUE means that either a motion job is active on the technology object or that the status has been set by the corresponding MCL instruction. If an axis or a kinematics is to be used in the PLC program with PLCopen instructions and at the same time with the Interpreter, the access can be synchronized by the instruction "setControlledByInterpreter".

Status of the axis:

Status	Description
<TO>.StatusInterpreterMotion.StatusWord.X0 (ControlledByInterpreter)	• 0 (FALSE): No control of the axis by the Interpreter technology object (no MCL instruction is active). • 1 (TRUE): Control of the axis by the Interpreter technology object (an MCL instruction is active or the bit has been set permanently by "setControlledByInterpreter"). If a PLCopen instruction becomes active on the axis, an alarm is triggered at the Interpreter technology object, causing the Interpreter technology object and thus the execution of the MCL program to be aborted.

Status of the connected kinematics:

Status	Description
<TO>.StatusInterpreterMotion.StatusWord.X0 (ControlledByInterpreter)	• 0 (FALSE): No control of the kinematics by the Interpreter technology object (no MCL instruction is active). • 1 (TRUE): Control of the kinematics by the Interpreter technology object (an MCL instruction is active or the bit has been set permanently by "setControlledByInterpreter"). If a PLCopen instruction becomes active on the kinematics, an alarm is triggered at the Interpreter technology object, causing the Interpreter technology object and thus the execution of the MCL program to be aborted.

Applies to

- Connected kinematics
- Single axis

Syntax

MCL

```
setControlledByInterpreter( [ obj := <val> ] [ ,ctrl := <val> ] );
```

Parameter	Data type	Default value	Description	
obj	AXIS_OBJECT	NULL	Axis instance[1)]	
ctrl	BOOL	TRUE	Value of status bit to be set[1)]	
			FALSE	Setting the value of status Bit ControlledByInterpreter to FALSE
			TRUE	Setting the value of status Bit ControlledByInterpreter to TRUE

1) Optional

Rules

- If the optional parameter "obj" is not specified or has the value ZERO, the instruction relates to the connected kinematics.
- When the Bit <TO>.StatusInterpreterMotion.StatusWord.X0 (ControlledByInterpreter) of a kinematics is set to TRUE, the system automatically sets the Bit <TO>.StatusInterpreterMotion.StatusWord.X0 (ControlledByInterpreter) of the axes associated with the kinematics to TRUE.
- Reading the status bit is possible from PLC and Interpreter side.

Example

MCL

```
// set ControlledByInterpreter at connected kinematics and linked axes
setControlledByInterpreter();
// same behaviour as previous command
setControlledByInterpreter( ctrl := TRUE );
// reset ControlledByInterpreter at connected kinematics and linked axes
setControlledByInterpreter( ctrl := FALSE );
// set ControlledByInterpreter at a single axis
setControlledByInterpreter( obj := myAxis );
// same behaviour as previous command
setControlledByInterpreter( obj := myAxis, ctrl := TRUE );
// reset ControlledByInterpreter at a single axis
setControlledByInterpreter( obj := myAxis, ctrl := FALSE );
```

8.6.5 setOvr() Set program override (S7-1500T)

Description

Use the MCL instruction "setOvr()" to modally set the program override factor, which acts on kinematics and single axis motions as a percentage factor for velocity and acceleration.

The program override factor is taken into account during the preparation of the Interpreter program.

The last valid value set in the program code is used for the program override factor. When the program is loaded, the modal value is initialized with the configuration value stored under "Extended parameters > Program preparation" (<TO_Interpreter>.Parameter.ProgramOverride). If the value is not set in the program code, the initialized value is used.

To prevent override changes from changing the original path, the program override acts initially on the axis velocities. The acceleration of the axes is adapted to the resulting velocities. The velocity override values that are set in the technology object data blocks of the axes or kinematics are also active and affect the dynamic responses of the active motions.

You can use this instruction in your MCL program for commissioning to track a motion (e.g. at the transition point) with reduced velocity and acceleration.

Applies to

- Kinematics
- Single axis

Influence on other MCL instructions

The modal parameters apply to the following MCL instructions:

- linAbs()
- linRel()
- circAbs()
- circRel()
- ptpAbs()
- ptpRel()
- ptpJtAbs()
- ptpJtRel()
- ptpAxAbs()
- ptpAxRel()
- move()
- posAbs()
- posRel()

Syntax

MCL

```
setOvr( <override> );
```

Parameters

The following table shows the parameters of the instruction "setOvr":

Parameter	Data type	Default value	Description	
override	LREAL	-	Modal value of the program override factor.	
			< 1.0	Not permitted
			1.0 ≤ override ≤ 100.0	Percentage default of the program override.
			> 100.0	Not permitted

Example

The following example shows how to set the program override:

MCL

```
// set dynamics modally
setDyn( v := 100.0, a := 5000.0, d := 6000.0, j := 10000.0 );
// linear movement using dynamics which were set by previous command
linAbs ( ( x := 20.0, y := 30.0, z := 40.0 ) );

// set program override to 50%
setOvr( 50.0 );
// linear movement with override 50% (v = 50.0, a = 2500.0, d = 3000.0)
linAbs ( ( x := 45.0 ) );
// set program override to 30%
setOvr( 30.0 );
// linear movement with override 30% (v = 15.0, a = 1500.0, d = 1800.0)
linAbs ( ( x := 75.0 ), v := 50.0 );
```

8.6.6 preHalt() Stop program preparation (S7-1500T)

Description

The Interpreter technology object has the following processing tasks:

- Interpretation of the Interpreter program instructions
- Preparing the motions for the Kinematics technology object and Axis technology object in advance (program preparation)
- Technology/time-optimized execution of the Interpreter jobs (program execution)

The preparation job can be stopped and triggered either selectively or conditionally. In the program preparation job, instructions are prepared and stored in the Interpreter job sequence. When the Interpreter job sequence is full, program preparation stops. The instructions from the Interpreter job sequence are processed in the program execution.

As soon as an instruction from the Interpreter job sequence has been processed, there is space in the Interpreter job sequence again. Program preparation prepares the next instruction and places it in the Interpreter job sequence. Since the program preparation supplies instructions during the execution of a motion sequence in the program execution, the number of instructions of a motion sequence is not limited by the size of the Interpreter job sequence.

One of the possibilities of explicit stopping of interpretation and motion preparation in program preparation by the user is to use the special "preHalt" system function in the Interpreter program. The "preHalt" function stops the preparation job.

After synchronization of preparation and program execution, preparation continues automatically, i.e. preparation of further instructions takes place after completion of all previous instructions.

Can be used in

- Program organization unit "Main program" of the Interpreter program
- Program organization unit "Function" of the Interpreter program

Syntax

MCL

```
preHalt();
```

Parameters

Parameter	Declaration	Data type	Default value	Description
None	-	-	-	-

Rules

- Stopping the interpretation and preparation in the program preparation with the "preHalt" function always results in stopping the motion as well.
- Blending of a motion is not possible using the "preHalt" function.
- The "preHalt" function is required to synchronize the execution and preparation tasks of the Interpreter program.

Example

In the following example, the "preHalt" function stops further preparation of instructions. Preparation of instructions after "preHalt" occurs after all previous instructions have been completed. Accordingly, the first linAbs function is executed with an exact stop ("velocity" = 0), although the first linAbs function was programmed with blending ("trans" = 1). After the first linAbs function is executed, the second linAbs function is prepared and executed. Assigning the value to "myVar1" is executed by using "preHalt()" in the program preparation beforehand.

MCL

```
linAbs( pos1, trans := 1, blend := 2 ); // polynomial blending
preHalt();
// Preprocessing evaluates further instructions only after all
// preceding instructions have been completed.
myVar1 := 5.1; // the value is not assigned in preprocessing because of preHalt()
linAbs( pos2, trans := 0 ); // exact stop
```

8.6.7 Time control of reading and writing variables (S7-1500T)

Reading and writing of program variables and technology object data block variables is performed by default as part of the preparation job.

In addition, it is possible to control the timing of read and write access to program variables and technology object data block variables

Time control of reading variables

Reading of technology object data block variables can occur in different contexts. The following table shows the different contexts of reading technology object data block variables in the MCL program:

Context of reading	Example

Reading of technology object data block variables in the context of preparing the Interpreter program	`...`
	`// $A1.ActualPosition is 0.0 in preprocessing`
	`posAbs( $A1, 20.0 );`
	`posAbs( $A1, 40.0 );`
	`...`
	`(* reading value of "ActualPosition" at the time of preprocessing of the assignment *)`
	`myVar := $A1.ActualPosition; // myVar := 0.0`
	`...`

Reading technology object data block variables after synchronization of program preparation/program execution by a previous preHalt function	`...`
	`posAbs( myExtAxis, 20.0 );`
	`...`
	`preHalt();  // Stop preprocessing`
	`myVar := myExtAxis.ActualPosition;  // myVar := 20.0`
	`...`

Time control of writing variables

Context of writing	Example

Writing technology object data block variables in the context of an assignment operation performed during preparation of the Interpreter program	`...`
	`posAbs( $A1, 20.0 );`
	`posAbs( $A1, 40.0 );`
	`...`
	`// executed in preprocessing`
	`$IPR.Clipboard.cbBool[1] := TRUE;`
	`...`

Writing of technology object data block variables after synchronization of program preparation/program execution by a previous preHalt function	`...`
	`posAbs( $A1, 20.0 );`
	`...`
	`preHalt(); // stop preprocessing`
	`...`
	`(* assigment value "0" to variable "cbDint[1]" executed in preprocessing, but after finishing the posAbs command because of the preHalt command. Without preHalt command the assigment would already take place before posAbs execution *)`
	`$IPR.Clipboard.cbDint[1] := 0;`
	`...`

Writing of technology object data block variables in the context of program execution by using the writeVar function	`...`
	`writeVar( $IPR.Clipboard.cbDint[1], 2 ); // execute in main run`
	`...`

When variables are written during program execution, different scenarios are possible regarding the time of execution:

- Execution within sequential program processing (example 1)
- Parallel execution to running instructions (see section "ON_START: Simultaneous start of instructions (Page 134)"):
  - Synchronous start with motion job (example 2)
  - Synchronous start in the blending point of a motion sequence (example 3)
  - Execution within the frame of the synchronous action

Example 1

In this example, the $IPR.Clipboard.cbDint[1] variable is set to 1 during preparation of the Interpreter program. The assignment is made before the program execution reaches the previous function. In the program execution, the writeVar function sets the variable $IPR.Clipboard.cbDint[1] from value 1 (value after the preparation loop) to value 2 after the motion to pos2 has completed.

MCL

```
linAbs( pos1, trans := 1, blend := 2 );
linAbs( pos2, trans := 0 );
writeVar( $IPR.Clipboard.cbDint[1], 2 ); // execute in main run
$IPR.Clipboard.cbDint[1] := 1; // execute in preprocessing
```

Example 2

This example shows the execution of the writeVar function (writing the variable $IPR.Clipboard.cbDint[1]) synchronously with the start of the "posAbs" function. After executing all instructions in the SYNC/END_SYNC block, the variable $IPR.Clipboard.cbDint[1] has the value 2.

MCL

```
$IPR.Clipboard.cbDint[1] := 0; // execute in preprocessing
SYNC
  posAbs( $A1, 20.0 );
ON_START
  writeVar( $IPR.Clipboard.cbDint[1], 2 ); // execute in main run
END_SYNC;
$IPR.Clipboard.cbDint[1] := 1; // execute in preprocessing
```

Example 3

In this example, the $IPR.Clipboard.cbDint[1] variable is set to the value 1 in the preparation of the Interpreter program. Then the variable $IPR.Clipboard.cbDint[1] is set to the value 2 in the program execution, at the blending point of the linear motion from pos1 to pos2.

MCL

```
$IPR.Clipboard.cbDint[1] := 1; // execute in preprocessing

linAbs( pos1, trans := 1, blend := 2 );
SYNC
  linAbs( pos2, trans := 0 );
ON_START
  writeVar( $IPR.Clipboard.cbDint[1], 2 ); // execute in main run
END_SYNC;
```

8.7 Bit string instructions (S7-1500T)

Description

Each Bitstring default function has two input parameters, denoted IN and N. The Bitstring function allows the bit pattern to be rotated or offset n places to the left or right. The input parameters to be shifted can be of DWORD type. The result of the instruction is returned in the operand as a function value. The following table shows the function names and data types of the two input parameters and the data type of the function value.

Syntax

Explanation of the input parameters:

- Input parameter IN: Buffer in which Bitstring operations are executed.
- Input parameter N: Number of cycles of the cyclic buffer function ROL and ROR or the number of digits to be shifted in case of SHL and SHR.

The type of the input parameter determines the type of the function result.

Function name	Data type of the input parameters (IN)	Data type of the input parameter (N)	Data type of the function value (RESULT)	Description
ROL	DWORD	UDINT	DWORD	The value in parameter IN is rotated left by the number of bit positions specified in the content of parameter N to the left.
ROR	DWORD	UDINT	DWORD	The value in parameter IN is rotated around the number of bit positions specified in the content of parameter N to the right.
SHL	DWORD	UDINT	DWORD	The value in parameter IN is shifted to the left by exactly the same number of digits as bit positions on the right side were replaced by 0, specified in parameter N.
SHR	DWORD	UDINT	DWORD	The value in parameter IN is shifted to the right by exactly the same number of digits as bit positions on the left side were replaced by 0, specified in parameter N.

Example

Function call	Result
RESULT := ROL (IN := 2#1101_0011, N := 5);	2#0111_1010 (= 122 decimal)
RESULT := ROR (IN := 2#1101_0011, N := 2);	2#1111_0100 (= 244 decimal)
RESULT := SHL (IN := 2#1101_0011, N := 3);	2#1001_1000 (= 152 decimal)
RESULT := SHR (IN := 2#1101_0011, N := 2);	2#0011_0100 (= 52 decimal)

8.8 Math functions (S7-1500T)

Description

Mathematical functions perform calculations based on input values (input parameters) and return numerical values.

Syntax

The following table describes a group of predefined mathematical functions together with function names and data types:

Function name	Data type of the input parameter (IN)	Data type of the function value (RESULT)	Description
ABS	DINT, LREAL	DINT, UDINT, LREAL	Absolute value
SQR	ANY_INT, LREAL	LREAL	Square
SQRT	ANY_INT, LREAL	LREAL	Square root
SIN	ANY_INT, LREAL	LREAL	Sine in arch
COS	ANY_INT, LREAL	LREAL	Cosine in arch
TAN	ANY_INT, LREAL	LREAL	Tangent in arch
EXP	ANY_INT, LREAL	LREAL	Exponential function
ASIN	ANY_INT, LREAL	LREAL	Arcsine
ACOS	ANY_INT, LREAL	LREAL	Arccosine
ATAN	ANY_INT, LREAL	LREAL	Arctangent
LN	ANY_INT, LREAL	LREAL	Natural logarithm
FRAC	ANY_INT, LREAL	LREAL	Returns the fractional part of the value

The abbreviations have the following meaning:

ANY_INT for data types DINT, UDINT

Function name	Data type of the input parameter (IN)		Data type of the function value (RESULT)	Description
MIN	IN1	ANY_INT, LREAL	ANY_INT, LREAL	Sets the minimum from min. 2 to max. 32 values. • IN1 – First input value • IN2 – Second input value • INn – Additionally inserted input variables whose values are to be compared
	IN2	ANY_INT, LREAL		
	INn	ANY_INT, LREAL		
MAX	IN1	ANY_INT, LREAL	ANY_INT, LREAL	Sets the maximum from min. 2 to max. 32 values. • IN1 – First input value • IN2 – Second input value • INn – Additionally inserted input variables whose values are to be compared
	IN2	ANY_INT, LREAL		
	INn	ANY_INT, LREAL		
LIMIT	MN	ANY_INT, LREAL	ANY_INT, LREAL	A transferred value is limited to a certain minimum or maximum value. • MN - Low limit • IN - Input value • MX - High limit
	IN	ANY_INT, LREAL		
	MX	ANY_INT, LREAL		

The abbreviations have the following meaning:

ANY_INT for data types DINT, UDINT

Example

Function call	Result
RESULT := ABS(-5);	5
RESULT := SQRT(81.0);	9.0 (LREAL)
RESULT := SQR(23);	529.0 (LREAL)
RESULT := EXP(4.1);	60.340 ...
RESULT := LN(2.718281);	0.99999.....
PI := 3.141592; RESULT := SIN(PI / 6);	0.5
RESULT := ACOS(0.5);	1.047197
RESULT := FRAC(34536.7);	0.7
RESULT := MIN(in1 := 2, in2 := 255, in3 := -4);	-4
RESULT := MAX(in1 := 2, in2 := 255, in3 := -4) ;	255
RESULT := LIMIT(mn := 0, in := 10, mx := 300);	10
RESULT := LIMIT(mn := 0, in := -10, mx := 300);	0
RESULT := LIMIT(mn := 0, in := 500, mx := 300);	300

8.9 Conversions (S7-1500T)

Description

The following table describes a group of predefined conversion functions together with their function name and data type.

Syntax

Function name	Data type of the input parameter (IN)		Data type of the function value (RESULT)	Description
FLOOR	LREAL		DINT	Rounds a value to the next lower integer.[1)]
CEIL	LREAL		DINT	Rounds a value up to the next higher integer[1)]
ROUND	LREAL		DINT	Rounds a value to the nearest integer. If the input value is exactly between an even and odd number, the even number is selected[1)]
TRUNC	LREAL		DINT	Selects the integer part and returns it as without the decimal places[1)]
SCALE_X	MIN	ANY_INT LREAL	ANY_INT LREAL	Scales a value based on the following equation: RESULT = [VALUE * (MAX - MIN)] + MIN • MIN – Low limit of the value range • MAX – High limit of the value range • VALUE – Value to be scaled
	VALUE	ANY_INT LREAL		
	MAX	ANY_INT LREAL		
NORM_X	MIN	LREAL	LREAL	Normalizes a value based on the following equation: RESULT = (VALUE – MIN) / (MAX – MIN) • MIN – Low limit of the value range • MAX – High limit of the value range • VALUE – Value to be normalized
	VALUE	LREAL		
	MAX	LREAL		

The abbreviations have the following meaning:

ANY_INT for data types DINT, UDINT

1) The behavior described applies to the value range -2147483648.0 ... 2147483647.0 of the floating-point number at the input parameter. The following applies outside this value range: If the floating-point number target is < -2147483648.0, the function value -2147483648 is output. If the floating-point number target is > 2147483647.0, the function value 2147483647 is output.

Example

Function call	Result
RESULT := FLOOR(5.34);	5
RESULT := FLOOR(-5.34);	-6
RESULT := CEIL(5.34);	6
RESULT := CEIL(-5.34);	-5
RESULT := ROUND(-7.64);	-8
RESULT := ROUND(-7.14);	-7
RESULT := TRUNC(-7.64);	-7
RESULT := SCALE_X(min := 5, value := 25, max := 105);	RESULT = [25 * (105 - 5)] + 5 = 2505.0
RESULT := NORM_X(min := 5, value := 25, max := 105);	RESULT = (25 – 5) / (105 – 5) = 0.2

See also

Data type conversions (Page 92)

Diagnostics (S7-1500T) 9

The description of Motion Control diagnostics is limited to the diagnostics view of the technology objects in TIA Portal, the technology alarms and the error IDs on Motion Control instructions.

"Debug" program mode is available for testing the execution of your Interpreter program.

The following descriptions can be found in the "S7-1500/S7-1500T Motion Control alarms and error IDs" documentation (Page 10):

- Diagnostics concept
- Technology alarms
- Error IDs in Motion Control instructions

A comprehensive description of the system diagnostics of the S7-1500 CPU can be found in the "Diagnostics" function manual (https://support.industry.siemens.com/cs/ww/en/view/59192926).

9.1 Interpreter technology object (S7-1500T)

9.1.1 Status and error bits (S7-1500T)

You can use the "Technology object > Diagnostics > Status and error bits" diagnostic function in the TIA Portal to monitor the status and error messages for the technology object. The diagnostics function is available in online operation.

The meaning of the status and error messages is described in the following tables. The associated technology object tag is given in parentheses.

Interpreter status

The following table shows possible states of the Interpreter technology object:

Status	Description
Active	The technology object is in operation. (<TO>.StatusWord.X0 (Control))
Error	The technology object has been enabled. You can move the axis with motion jobs. (<TO>.StatusWord.X1 (Error))
Restart active	The technology object will be reinitialized. (<TO>.StatusWord.X2 (RestartActive))
Master control active	The toolbar of the programming editor has master control over the technology object. You can control the Interpreter conditionally from the user program. (<TO>.StatusWord.X4 (MasterControlActive))
Restart required	Data relevant for the restart has been changed. The changes are applied only after a restart of the technology object. (<TO>.StatusWord.X3 (OnlineStartValuesChanged))

Interpreter program status

The following table shows possible states of the Interpreter program in the Interpreter technology object:

Status	Description
Loading	The technology object is loading the Interpreter program. Interpreter program preparation is running. (<TO>.StatusWord.X9 (Loading))
Loaded	The Interpreter program is loaded and prepared. (<TO>.StatusWord.X10 (Loaded))
Running	The technology object is executing an Interpreter program. (<TO>.StatusWord.X5 (InRun))
Done (no job running)	Execution of the Interpreter program is complete. (<TO>.StatusWord.X6 (Done))

Status	Description
Stop	Execution of the Interpreter program will be or has been stopped. (<TO>.StatusWord.X7 (Stopping))
Interrupted	Execution of the Interpreter program was interrupted and can be continued. (<TO>.StatusWord.X8 (Interrupted))
Debug program mode active	Debug program mode is active. (<TO>.StatusInterpreter.ProgramMode = 2))

Errors

The following table shows the possible errors:

Error	Description
System	A system-internal error has occurred. (<TO>.ErrorWord.X0 (SystemFault))
Configuration	A configuration error has occurred. One or more configuration parameters are inconsistent or invalid. The technology object was incorrectly configured, or editable configuration data was incorrectly modified during runtime of the user program. (<TO>.ErrorWord.X1 (ConfigFault))
User program	An error occurred in the user program with a Motion Control instruction or its use. (<TO>.ErrorWord.X2 (UserFault))
Job rejected	A job cannot be executed. You cannot execute any Motion Control instructions because the necessary requirements have not been fulfilled. (<TO>.ErrorWord.X3 (CommandNotAccepted))
Current Interpreter program	An error has occurred in the current Interpreter program. (<TO>.ErrorWord.X4 (UserProgramFault))
Current mapping	An error occurred in the current Interpreter mapping (<TO>.ErrorWord.X5 (UserMappingFault))

Warnings

The following table shows possible warnings:

Warning	Description
System	A system-internal warning has occurred. (<TO>.WarningWord.X0 (SystemWarning))
Configuration	One or multiple configuration parameters are being temporarily internally adapted. (<TO>.WarningWord.X1 (ConfigWarning))
User program	A warning has occurred in the user program. (<TO>.WarningWord.X2 (UserWarning))

Warning	Description
Job rejected	Job cannot be executed. You cannot execute any Motion Control instructions because the necessary requirements have not been fulfilled. (<TO>.WarningWord.X3 (CommandNotAccepted))
Current Interpreter program	A warning has occurred in the current Interpreter program. (<TO>.WarningWord.X4 (UserProgrammWarning))
Current mapping	A warning has occurred in the current Interpreter mapping. (<TO>.WarningWord.X5 (UserMappingWarning))

Alarm display

For more information and to acknowledge the error, go to the Inspector window by clicking on the "Alarm display" link.

More information

An option for evaluating the individual status bits can be found in the section "Evaluating StatusWord, ErrorWord and WarningWord" of the "S7-1500/S7-1500T Motion Control overview" (Page 10) documentation.

9.1.2 Interpreter status (S7-1500T)

You can use the "Technology object > Diagnostics > Interpreter status" diagnostics function to monitor the status of the Interpreter in the TIA Portal. The diagnostics function is available in online operation.

"Currently loaded Interpreter program" area

The following table shows the meaning of the status information:

Status	Description
Program name	Name of the loaded Interpreter program <TO>.ProgramName
Program mode	Operating mode of interpreter program execution <TO>.StatusInterpreter.ProgramMode
Program status	Status of the loaded Interpreter program
Interpreter mapping	Name of the Interpreter mapping technology object used <TO>.MappingName

"Details about the error" area

The following table describes the meaning of the error information:

Status	Description
Program name	Name of the loaded Interpreter program <TO>.ProgramName
Line	Line number of the error that occurred <TO>.ErrorDetail.LineNumber
Error details	Additional information about the error <TO>.ErrorDetail.ErrorInfo

Instructions (S7-1500T) 10

10.1 Interpreter (S7-1500T)

10.1.1 MC_LoadProgram V10 (S7-1500T)

10.1.1.1 MC_LoadProgram: Load/unload Interpreter program V10 (S7-1500T)

Description

Use the Motion Control instruction "MC_LoadProgram" to load an Interpreter program into the Interpreter technology object. The interpreter prepares the interpreter program for execution.

To load a modified or different interpreter program into the interpreter technology object, first unload the loaded interpreter program from the interpreter technology object. You can then load the modified or new Interpreter program into the Interpreter technology object.

The Motion Control instruction offers you the following:

- Loading the Interpreter program into the Interpreter technology object (Page 51)
- Unloading the Interpreter program from the Interpreter technology object (Page 51)

Applies to

- Interpreter

Requirements

- The Interpreter technology object has been configured correctly.
- No Motion Control job is active at the Interpreter technology object ("<TO>.StatusWord.X0" = FALSE (Control)).

Override response

A new "MC_LoadProgram" job does not abort any active Motion Control job.

A "MC_LoadProgram" job is aborted by a "MC_StopProgram" job.

The override response for "MC_LoadProgram" jobs is described in section "Override response V10: Interpreter jobs (Page 358)".

Parameters

The following table shows the parameters of the Motion Control instruction "MC_LoadProgram":

Parameter	Declaration	Data type	Default value	Description	
Interpreter	INPUT	TO_Interpreter	-	Interpreter technology object	
Execute	INPUT	BOOL	FALSE	TRUE	Start job with a positive edge
Program	INPUT	STRING	-	When "Mode" = 0: Not relevant	
				When "Mode" = 1: Name of the technology object Interpreter program	
ProgramSource	INPUT	DINT	1	When "Mode" = 0: Not relevant	
				When "Mode" = 1: Source of the Interpreter program	
				1	Interpreter program technology object
				2	Reserved
Mapping	INPUT	STRING	-	When "Mode" = 0: Not relevant	
				When "Mode" = 1: Enter name of the Interpreter mapping technology object	
MappingSource	INPUT	DINT	1	When "Mode" = 0: Not relevant	
				When "Mode" = 1: Source of the Interpreter mappings	
				1	Interpreter mapping technology object
				2	Reserved
				3	Reserved
Mode	INPUT	DINT	1	Download mode	
				0	Unloading the Interpreter program from the Interpreter technology object
				1	Loading the Interpreter program into the Interpreter technology object The Interpreter program is disabled for changes.
Done	OUTPUT	BOOL	FALSE	TRUE	Job is completed.
Busy	OUTPUT	BOOL	FALSE	TRUE	The job is being processed.

Parameter	Declaration	Data type	Default value	Description	
CommandAborted	OUTPUT	BOOL	FALSE	TRUE	The job was aborted by another job during execution.
Error	OUTPUT	BOOL	FALSE	TRUE	An error occurred while processing the job. The job is rejected. The cause of the error can be found in the "ErrorID" parameter.
ErrorID	OUTPUT	WORD	16#0000	Error ID for parameter "ErrorID" You can find more detailed information in the "Error IDs" section of the "S7-1500/S7-1500T Motion Control alarms and error IDs" documentation (Page 10).	

10.1.2 MC_RunProgram V10 (S7-1500T)

10.1.2.1 MC_RunProgram: Start execution of the Interpreter program V10 (S7-1500T)

Description

Use the Motion Control instruction "MC_RunProgram" to start the execution of the Interpreter program that is being or has been loaded in the Interpreter technology object.

The Motion Control instruction offers you the following:

- Run Interpreter program (Page 52)

Applies to

- Interpreter

Requirement

- The Interpreter technology object has been configured correctly.
- The Interpreter program to be executed is loaded or has been loaded in the Interpreter technology object ("<TO>.StatusWord.X9" = TRUE (Loading) or "<TO>.StatusWord.X10" = TRUE (Loaded)).
- No other "MC_RunProgram" job is active on the Interpreter technology object.
- No "MC_Stop" job is active on the Interpreter technology object.

Override response

A new "MC_RunProgram" job does not abort any active Motion Control job.

A "MC_RunProgram" job is aborted by a "MC_StopProgram" job.

The override response for "MC_RunProgram" jobs is described in section "Override response V10: Interpreter jobs (Page 358)".

Parameters

The following table shows the parameters of the Motion Control instruction "MC_RunProgram":

Parameters	Declaration	Data type	Default value	Description	
Interpreter	INPUT	TO_Interpreter	-	Interpreter technology object	
Execute	INPUT	BOOL	FALSE	TRUE	Start job with a positive edge
Done	OUTPUT	BOOL	FALSE	TRUE	Job is completed. The Interpreter program was executed and prepared for another execution.
Busy	OUTPUT	BOOL	FALSE	TRUE	The job is being processed.
Active	OUTPUT	BOOL	FALSE	TRUE	The Interpreter program is running.
CommandAborted	OUTPUT	BOOL	FALSE	TRUE	The job was aborted by another job during execution.
Error	OUTPUT	BOOL	FALSE	TRUE	An error occurred while processing the job. The job is rejected. The cause of the error can be found in the "ErrorID" parameter.
ErrorID	OUTPUT	WORD	16#0000	Error ID for parameter "ErrorID" You can find more detailed information in the "Error IDs" section of the "S7-1500/S7-1500T Motion Control alarms and error IDs" documentation (Page 10).	

10.1.3 MC_StopProgram V10 (S7-1500T)

10.1.3.1 MC_StopProgram: Stop execution of Interpreter program V10 (S7-1500T)

Description

Use the "MC_StopProgram" Motion Control instruction to stop the execution of the Interpreter program in the Interpreter technology object. A single axis/kinematics controlled by the interpreter program is stopped depending on the specified mode ("<TO>.StatusInterpreterMotion.StatusWord.X0" = TRUE (ControlledByInterpreter)). The corresponding technology object of the single axis/kinematics is then no longer controlled by the Interpreter technology object.

NOTE

Never-ending motion jobs

Note that for motion jobs that do not end automatically, such as "move()", the "<TO>.StatusInterpreterMotion.StatusWord.X0" tag (ControlledByInterpreter) of the corresponding technology object is set to "FALSE" as soon as the technology object has reached the specified state.

To also cancel these motion jobs with a "MC_StopProgram" job, use the MCL instruction "setControlledByInterpreter()".

The Motion Control instruction offers you the following:

- Stop execution of Interpreter program (Page 53)

Applies to

- Interpreter

Requirement

- The Interpreter technology object has been configured correctly.
- No other "MC_StopProgram" job with the same or a higher priority dynamic mode is active on the Interpreter technology object.
- The "Debug" program mode does not have master control over the Interpreter technology object.

Override response

A new "MC_StopProgram" job with "Mode" = 0 aborts a current "MC_StopProgram" job with "Mode" = 1 or 2.

A new "MC_StopProgram" job with "Mode" = 1 aborts a current "MC_StopProgram" job with "Mode" = 2.

The override response for "MC_StopProgram" jobs is described in section "Override response V10: Interpreter jobs (Page 358)".

Parameters

The following table shows the parameters of the Motion Control instruction "MC_StopProgram":

Parameters	Declaration	Data type	Default value	Description	
Interpreter	INPUT	TO_Interpreter	-	Interpreter technology object	
Execute	INPUT	BOOL	FALSE	TRUE	Start job with a positive edge
Mode	INPUT	DINT	0	Mode for dynamic behavior	
				0	Stop single axis/kinematics with maximum dynamics
				1	Stop single axis/kinematics with the dynamics of the motion job to be interrupted
				2	Stop single axis/kinematics after the current motion job or after the current motion sequence.
Done	OUTPUT	BOOL	FALSE	TRUE	Job is completed. The execution of the Interpreter program has been stopped.
Busy	OUTPUT	BOOL	FALSE	TRUE	The job is being processed.
CommandAborted	OUTPUT	BOOL	FALSE	TRUE	The job was aborted by another job during execution.
Error	OUTPUT	BOOL	FALSE	TRUE	An error occurred while processing the job. The job is rejected. The cause of the error can be found in the "ErrorID" parameter.
ErrorID	OUTPUT	WORD	16#0000	Error ID for parameter "ErrorID" You can find more detailed information in the "Error IDs" section of the "S7-1500/S7-1500T Motion Control alarms and error IDs" documentation (Page 10).	

10.2 Override response of Motion Control jobs V10 (S7-1500T)

10.2.1 Override response V10: Homing and motion jobs (S7-1500T)

The following table shows how a new Motion Control job affects active homing and motion jobs:

⇒ **Active job** ⇓ **New job**	**MC_Home "Mode" = 2-, 8, 10**	**MC_Home "Mode" = 3, 5**	**MC_Halt "Mode" = 1 MC_Move-Absolute/ MC_Move-Relative/ MC_Position-Profile "Buffer-Mode" = 0, 1 active MC_Move-Velocity MC_MoveJog**	**MC_Move-Absolute/ MC_Move-Relative/ MC_Position-Profile "Buffer-Mode" = 1 waiting**	**MC_Halt "Mode" = 0**	**MC_Stop**	**MC_Move-Super-imposed MC_Motion-InSuper-imposed MC_Motion-InSuper-imposed-Axes MC_Halt-Super-imposed**	**MC_Motion-InVelocity MC_Motion-InPosition**
MC_Home[1] "Mode" = 3, 5	A	A	A	A	A	N	A	A
MC_Home "Mode" = 9	A	-	-	-	-	N	-	-
MC_Halt "Mode" = 1	-	A	A	A	-	N	-	A
MC_Halt "Mode" = 0 **MC_MoveAbsolute/ MC_MoveRelative/ MC_PositionProfile** "BufferMode" = 0 Active **MC_MoveVelocity MC_MoveJog**	-	A	A	A	A	N	A	A

A The running job is aborted with "CommandAborted" = TRUE.

B An "MC_Stop" job is aborted by another "MC_Stop" job with a stop response that is the same or higher.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) In the case of a virtual or simulated axis, the running job is not aborted.

2) The status "Busy" = TRUE, "StartSync" = FALSE, "InSync" = FALSE corresponds to a waiting synchronous operation.

3) The status "Busy" = TRUE, "StartSync" or "InSync" = TRUE corresponds to an active synchronous operation.

4) The status "Busy" = TRUE, "StartSyncOut" = FALSE corresponds to a pending desynchronization job.

5) The status "Busy" = TRUE, "StartSyncOut" = TRUE corresponds to an active desynchronization job.

6) An "MC_CamIn" job with "SyncProfileReference" = 5 does not abort an "MC_[...]Superimposed" job. Running job continues to be executed.

⇒ Active job ⇓ New job	MC_Home "Mode" = 2-, 8, 10	MC_Home "Mode" = 3, 5	MC_Halt "Mode" = 1 MC_MoveAbsolute/ MC_MoveRelative/ MC_PositionProfile "BufferMode" = 0, 1 active MC_MoveVelocity MC_MoveJog	MC_MoveAbsolute/ MC_MoveRelative/ MC_PositionProfile "BufferMode" = 1 waiting	MC_Halt "Mode" = 0	MC_Stop	MC_MoveSuperimposed MC_MotionInSuperimposed MC_MotionInSuperimposedAxes MC_HaltSuperimposed	MC_MotionInVelocity MC_MotionInPosition
MC_MotionInVelocity **MC_MotionInPosition**								
MC_MoveAbsolute/ MC_MoveRelative/ MC_PositionProfile "BufferMode" = 1 active, waiting	N	N	-/N	N	N	N	-	N
MC_MoveSuperimposed **MC_MotionInSuperimposed** **MC_MotionInSuperimposedAxes** **MC_HaltSuperImposed**	-	-	-	-	-	N	A	-
MC_Stop	A	A	A	A	A	B	A	A
MC_GearIn **MC_GearInVelocity**	-	A	A	A	A	N	A	A
MC_GearInPos **MC_CamIn** waiting[2)]	-	-	-	A	-	N	-	-
MC_GearInPos **MC_CamIn** active[3)]	-	A	A	A	A	N	A[6)]	A

A The running job is aborted with "CommandAborted" = TRUE.

B An "MC_Stop" job is aborted by another "MC_Stop" job with a stop response that is the same or higher.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) In the case of a virtual or simulated axis, the running job is not aborted.

2) The status "Busy" = TRUE, "StartSync" = FALSE, "InSync" = FALSE corresponds to a waiting synchronous operation.

3) The status "Busy" = TRUE, "StartSync" or "InSync" = TRUE corresponds to an active synchronous operation.

4) The status "Busy" = TRUE, "StartSyncOut" = FALSE corresponds to a pending desynchronization job.

⇒ Active job ⇓ New job	MC_Home "Mode" = 2-, 8, 10	MC_Home "Mode" = 3, 5	MC_Halt "Mode" = 1 MC_Move-Absolute/ MC_Move-Relative/ MC_Position-Profile "Buffer-Mode" = 0, 1 active MC_Move-Velocity MC_MoveJog	MC_Move-Absolute/ MC_Move-Relative/ MC_Position-Profile "Buffer-Mode" = 1 waiting	MC_Halt "Mode" = 0	MC_Stop	MC_Move-Super-imposed MC_Motion-InSuper-imposed MC_Motion-InSuper-imposed-Axes MC_Halt-Super-imposed	MC_Motion-InVelocity MC_Motion-InPosition
MC_Leading-ValueAdditive	-	-	-	-	-	-	-	-
MC_GearOut **MC_CamOut** waiting[4)]	N	N	N	N	N	N	-	N
MC_GearOut **MC_CamOut** active[5)]	N	N	N	N	N	N	A	N

A The running job is aborted with "CommandAborted" = TRUE.

B An "MC_Stop" job is aborted by another "MC_Stop" job with a stop response that is the same or higher.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) In the case of a virtual or simulated axis, the running job is not aborted.

2) The status "Busy" = TRUE, "StartSync" = FALSE, "InSync" = FALSE corresponds to a waiting synchronous operation.

3) The status "Busy" = TRUE, "StartSync" or "InSync" = TRUE corresponds to an active synchronous operation.

4) The status "Busy" = TRUE, "StartSyncOut" = FALSE corresponds to a pending desynchronization job.

5) The status "Busy" = TRUE, "StartSyncOut" = TRUE corresponds to an active desynchronization job.

6) An "MC_CamIn" job with "SyncProfileReference" = 5 does not abort an "MC_[...]Superimposed" job. Running job continues to be executed.

NOTE

Override response with active fixed stop

With an active force and torque limiting with "MC_TorqueLimiting", running jobs are aborted if the drive is held at the fixed stop with "InClamping" = TRUE.

10.2.2 Override response V10: Synchronous operation jobs (S7-1500T)

The following table shows how a new Motion Control job affects the motion of the axis on active synchronous operation jobs:

⇒ **Active job** ⇓ **New job**	**MC_Gear-In**	**MC_Gear-InVelocity**	**MC_Gear-InPos MC_Cam-In waiting**[2]	**MC_Gear-InPos MC_Cam-In active**[3]	**MC_Phasing-Absolute MC_Phasing-Relative**	**MC_Offset-Absolute MC_Offset-Relative**	**MC_Leading-ValueAdditive**	**MC_Gear-Out MC_Cam-Out waiting**[4]	**MC_Gear-Out MC_Cam-Out active**[5]
MC_Home[1] "Mode" = 3, 5	A	A	-	N	-	-	-	-	A
MC_Halt "Mode" = 0, 1 **MC_Move-Absolute/ MC_MoveRel-ative** "BufferMode" = 0 Active **MC_Position-Profile** "BufferMode" = 0 Active **MC_Move-Velocity** **MC_MoveJog**	A	A	-	A	A	A	-	A	A
MC_Move-Absolute/- MC_MoveRel-ative "BufferMode" = 1 active, waiting	N	N	-	N	N	N	-	N	N

A The running job is aborted with "CommandAborted" = TRUE.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) In the case of a virtual or simulated axis, the running job is not aborted.

2) A waiting synchronous operation job ("Busy" = TRUE, "StartSync" = FALSE, "InSync" = FALSE) does not abort any active jobs. Abort with an "MC_Power" job is possible.

3) The status "Busy" = TRUE, "StartSync" or "InSync" = TRUE corresponds to an active synchronous operation.

4) A pending desynchronization job ("Busy" = TRUE, "StartSyncOut" = FALSE) does not abort any active jobs. Abort with an "MC_Power" job is possible.

5) The status "Busy" = TRUE, "StartSyncOut" = TRUE corresponds to an active desynchronization job.

6) When the following axis is in position-controlled mode, execution of the running job is continued. When the following axis is not in position-controlled mode, the new job is rejected.

7) An "MC_GearOut" job only terminates an "MC_Gear[...]" job. Correspondingly, an "MC_CamOut" job only cancels a "MC_Cam[...]" job.

8) A job with "SyncProfileReference" = 5 aborts a pending synchronous operation. Canceling a pending synchronous operation has no influence on an active synchronous operation.

⇒ Active job ⇓ New job	MC_GearIn	MC_GearInVelocity	MC_GearInPos MC_CamIn waiting[2]	MC_GearInPos MC_CamIn active[3]	MC_PhasingAbsolute MC_PhasingRelative	MC_OffsetAbsolute MC_OffsetRelative	MC_LeadingValueAdditive	MC_GearOut MC_CamOut waiting[4]	MC_GearOut MC_CamOut active[5]
MC_PositionProfile "BufferMode" = 1 active, waiting									
MC_MotionInVelocity **MC_MotionInPosition**	A	A	-	A	A	A	-	A	A
MC_MoveSuperimposed **MC_MotionInSuperimposed** **MC_MotionInSuperimposedAxes** **MC_HaltSuperImposed**	-	-/N[6]	-	-	-	-	-	-	-
MC_Stop	A	A	A	A	A	A	-	A	A
MC_GearIn **MC_GearInVelocity**	A	A	A	A	A	A	-	A	A
MC_GearInPos **MC_CamIn** waiting[2]	-	-	A	-	-	-	-	A	-

A The running job is aborted with "CommandAborted" = TRUE.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) In the case of a virtual or simulated axis, the running job is not aborted.

2) A waiting synchronous operation job ("Busy" = TRUE, "StartSync" = FALSE, "InSync" = FALSE) does not abort any active jobs. Abort with an "MC_Power" job is possible.

3) The status "Busy" = TRUE, "StartSync" or "InSync" = TRUE corresponds to an active synchronous operation.

4) A pending desynchronization job ("Busy" = TRUE, "StartSyncOut" = FALSE) does not abort any active jobs. Abort with an "MC_Power" job is possible.

5) The status "Busy" = TRUE, "StartSyncOut" = TRUE corresponds to an active desynchronization job.

6) When the following axis is in position-controlled mode, execution of the running job is continued. When the following axis is not in position-controlled mode, the new job is rejected.

7) An "MC_GearOut" job only terminates an "MC_Gear[...]" job. Correspondingly, an "MC_CamOut" job only cancels a "MC_Cam[...]" job.

8) A job with "SyncProfileReference" = 5 aborts a pending synchronous operation. Canceling a pending synchronous operation has no influence on an active synchronous operation.

⇒ **Active job** ⇓ **New job**	**MC_GearIn**	**MC_GearInVelocity**	**MC_GearInPos MC_CamIn waiting**[2]	**MC_GearInPos MC_CamIn active**[3]	**MC_PhasingAbsolute MC_PhasingRelative**	**MC_OffsetAbsolute MC_OffsetRelative**	**MC_LeadingValueAdditive**	**MC_GearOut MC_CamOut waiting**[4]	**MC_GearOut MC_CamOut active**[5]
MC_GearInPos MC_CamIn active[3]	A	A[6]	A	A	A	A	-	A	A
MC_PhasingAbsolute MC_PhasingRelative	-	N	-	-	A	N	-	-	N
MC_OffsetAbsolute MC_OffsetRelative	-	N	-	-	N	A	-	-	N
MC_LeadingValueAdditive	-	-	-	-	-	-	A	-	-
MC_GearOut MC_CamOut waiting[4]	-	N	A[7] [8]	-	-	-	-	A[7]	N
MC_GearOut MC_CamOut active[5]	A[7]	N	A[7] [8]	A[7]	A	A	-	A[7]	N

A The running job is aborted with "CommandAborted" = TRUE.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) In the case of a virtual or simulated axis, the running job is not aborted.

2) A waiting synchronous operation job ("Busy" = TRUE, "StartSync" = FALSE, "InSync" = FALSE) does not abort any active jobs. Abort with an "MC_Power" job is possible.

3) The status "Busy" = TRUE, "StartSync" or "InSync" = TRUE corresponds to an active synchronous operation.

4) A pending desynchronization job ("Busy" = TRUE, "StartSyncOut" = FALSE) does not abort any active jobs. Abort with an "MC_Power" job is possible.

5) The status "Busy" = TRUE, "StartSyncOut" = TRUE corresponds to an active desynchronization job.

6) When the following axis is in position-controlled mode, execution of the running job is continued. When the following axis is not in position-controlled mode, the new job is rejected.

7) An "MC_GearOut" job only terminates an "MC_Gear[...]" job. Correspondingly, an "MC_CamOut" job only cancels a "MC_Cam[...]" job.

8) A job with "SyncProfileReference" = 5 aborts a pending synchronous operation. Canceling a pending synchronous operation has no influence on an active synchronous operation.

NOTE

Override response with active fixed stop

With an active force and torque limiting with "MC_TorqueLimiting", running jobs are aborted if the drive is held at the fixed stop with "InClamping" = TRUE.

10.2.3 Override response V10: Measuring input jobs (S7-1500T)

The following table shows which new Motion Control jobs will override active measuring input jobs:

⇒ **Active job** ⇓ **New job**	**MC_MeasuringInput**	**MC_MeasuringInputCyclic**
MC_Home "Mode" = 2, 3, 5, 8, 9, 10	B	B
MC_Home "Mode" = 0, 1, 6, 7, 11, 12	-	-
MC_MeasuringInput **MC_MeasuringInputCyclic** **MC_AbortMeasuringInput**	A	A

A The running job is aborted with "CommandAborted" = TRUE.

B The running job is aborted with "ErrorID" = 16#80A3.

- No effect. Running job continues to be executed.

10.2.4 Override response V10: Kinematics motion commands (S7-1500T)

Single axis jobs are not overridden by kinematics jobs.

The following table shows how a new Motion Control job affects active kinematics motion jobs:

⇒ **Active job** ⇓ **New job**	**MC_MoveLinearAbsolute** **MC_MoveLinearRelative** **MC_MoveCircularAbsolute** **MC_MoveCircularRelative** **MC_MoveDirectAbsolute** **MC_MoveDirectRelative** **MC_TrackConveyorBelt** **MC_DefineWorkspaceZone** **MC_DefineKinematicsZone** **MC_SetWorkspaceZoneActive** **MC_SetWorkspaceZoneInactive** **MC_SetKinematicsZoneActive** **MC_SetKinematicsZoneInactive** **MC_SetOcsFrame**	**MC_GroupInterrupt**	**MC_GroupStop**
MC_Home **MC_MoveSuperimposed** **MC_GearOut** **MC_CamOut**	N	N	N
MC_Halt **MC_MoveAbsolute/** **MC_MoveRelative/** **MC_PositionProfile** "BufferMode" = 0 Active **MC_MoveVelocity** **MC_MoveJog** **MC_Stop** **MC_GearIn** **MC_GearInPos** **MC_GearInVelocity** **MC_CamIn** **MC_MotionInVelocity** **MC_MotionInPosition**	A	A	A
MC_GroupStop	A	A	N

A The running job is aborted with "CommandAborted" = TRUE.

B Running job is interrupted or resumed.

C Synchronization of the OCS with the conveyor belt is aborted with "MC_SetOcsFrame" = TRUE.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed. A new kinematics job is added to the job sequence.

⇒ **Active job** ⇓ **New job**	**MC_MoveLinearAbsolute** **MC_MoveLinearRelative** **MC_MoveCircularAbsolute** **MC_MoveCircularRelative** **MC_MoveDirectAbsolute** **MC_MoveDirectRelative** **MC_TrackConveyorBelt** **MC_DefineWorkspaceZone** **MC_DefineKinematicsZone** **MC_SetWorkspaceZoneActive** **MC_SetWorkspaceZoneInactive** **MC_SetKinematicsZoneActive** **MC_SetKinematicsZoneInactive** **MC_SetOcsFrame**	**MC_GroupInterrupt**	**MC_GroupStop**
MC_GroupInterrupt **MC_GroupContinue**	B	A	N
MC_MoveLinearAbsolute **MC_MoveLinearRelative** **MC_MoveCircularAbsolute** **MC_MoveCircularRelative** **MC_MoveDirectAbsolute** **MC_MoveDirectRelative** **MC_TrackConveyorBelt** **MC_DefineWorkspaceZone** **MC_DefineKinematicsZone** **MC_SetWorkspaceZoneActive** **MC_SetWorkspaceZoneInactive** **MC_SetKinematicsZoneActive** **MC_SetKinematicsZoneInactive**	-	-	N
MC_SetOcsFrame	C, -	-	N

A The running job is aborted with "CommandAborted" = TRUE.

B Running job is interrupted or resumed.

C Synchronization of the OCS with the conveyor belt is aborted with "MC_SetOcsFrame" = TRUE.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed. A new kinematics job is added to the job sequence.

10.2.5 Override response V10: Interpreter jobs (S7-1500T)

The following table shows how a new Motion Control job affects running Interpreter jobs:

⇒ **Active job** ⇓ **New job**	**MC_LoadProgram**	**MC_RunProgram**	**MC_StopProgram**
MC_LoadProgram	N	N	N
MC_RunProgram	-[1)]	N	N
MC_StopProgram	A	A	A[2)]
MC_Reset	-/N[3)]	-/N[3)]	-/N[3)]

A The running job is aborted with "CommandAborted" = TRUE.

N Not permitted. Running job continues to be executed. The new job is rejected.

- No effect. Running job continues to be executed.

1) The new job is also executed.

2) A new "MC_StopProgram" job with "Mode" = 0 cancels a running "MC_StopProgram" job with "Mode" = 1 or 2. A new "MC_StopProgram" job with "Mode" = 1 aborts a current "MC_StopProgram" job with "Mode" = 2.

3) An "MC_Reset" job with "Restart" = TRUE is rejected. Running job continues to be executed.

Override response for technology objects controlled by the Interpreter

The following table shows how a new Motion Control job acts in relation to a technology object controlled by the Interpreter during the execution of an Interpreter program ("<TO>.StatusInterpreterMotion.StatusWord.X0" = TRUE (ControlledByInterpreter)):

⇓ **New job**	**The new job will be executed.**	**The new job aborts the execution of the Interpreter program.**
MC_Power disable/enable with technology object	✓	✓
MC_Reset with "Restart" = TRUE	✓	✓
MC_Reset with "Restart" = FALSE	✓	-
MC_Home **MC_Halt** **MC_Stop**	✓	✓
MC_MoveAbsolute **MC_MoveRelative** **MC_MoveVelocity** **MC_MoveJog**	✓	✓
MC_SetAxisSTW **MC_WriteParameter**	✓	-
MC_MoveSuperimposed **MC_MotionInSuperimposed** **MC_MotionInSuperimposedAxes** **MC_HaltSuperimposed**	-	-
MC_MotionInVelocity **MC_MotionInPosition**	✓	✓

1) When the synchronization/desynchronization starts, the execution of the Interpreter program is cancelled.

2) If a technology alarm is issued due to a fixed stop, the execution of the Interpreter program is cancelled.

⇓ New job	The new job will be executed.	The new job aborts the execution of the Interpreter program.
MC_GearIn **MC_GearInPos** **MC_GearInVelocity** **MC_GearOut** **MC_CamIn** **MC_CamOut**	✓	✓[1)]
MC_PhasingRelative **MC_PhasingAbsolute**	-	-
MC_LeadingValueAdditive	✓	-
MC_OffsetRelative **MC_OffsetAbsolute**	-	-
MC_TorqueAdditive **MC_TorqueRange**	✓	-
MC_TorqueLimiting	✓	-/✓[2)]
MC_SynchronizedMotionSimulation	✓	-
MC_MoveLinearAbsolute **MC_MoveLinearRelative** **MC_MoveCircularAbsolute** **MC_MoveCircularRelative** **MC_MoveDirectAbsolute** **MC_MoveDirectRelative**	-	-
MC_GroupInterrupt **MC_GroupContinue**	-	-
MC_GroupStop	✓	✓
MC_KinematicsMotionSimulation	✓	-
MC_TrackConveyorBelt	-	-
MC_DefineTool **MC_SetTool** **MC_SetOcsFrame**	-	-
MC_KinematicsTransformation **MC_InverseKinematicsTransformation**	✓	-
MC_DefineWorkspaceZone **MC_SetWorkspaceZoneActive** **MC_SetWorkspaceZoneInactive** **MC_DefineKinematicsZone** **MC_SetKinematicsZoneActive** **MC_SetKinematicsZoneInactive**	-	-

1) When the synchronization/desynchronization starts, the execution of the Interpreter program is cancelled.

2) If a technology alarm is issued due to a fixed stop, the execution of the Interpreter program is cancelled.

NOTE

MC_Power

Note that a "MC_Power" request with "Enable" = FALSE always locks the specified technology object, even if the technology object was enabled via an Interpreter program with "powerOn()".

Mixed operation of MC_Power (SCL) and powerOn() (MCL) is not permitted.

NOTE

MC_Stop

Note that when starting a motion controlled by the Interpreter no "MC_Stop" job may be active, otherwise the motion job will be aborted.

NOTE

MC_GroupStop

Note that when starting a kinematics motion controlled by the Interpreter, no "MC_GroupStop" job may be active, otherwise the motion job will be aborted.

A "MC_GroupStop" job only affects active kinematics motions or if the kinematics technology object is controlled by the Interpreter ("<TO>.StatusInterpreterMotion.StatusWord.X0" = TRUE (ControlledByInterpreter)).

See also

Override response V10: Homing and motion jobs (Page 349)

Override response V10: Synchronous operation jobs (Page 352)

Override response V10: Kinematics motion commands (Page 356)

Tags of the technology object data blocks (S7-1500T) 11

11.1 Legend (S7-1500T)

Tag	Name of the tag	
Data type	Data type of the tag	
Values	Value range of the tag - minimum value to maximum value (L - linear specification R - rotary specification) If no specific value is shown, the value range limits of the relevant data type apply or the information under "Description".	
W	Effectiveness of changes in the technology data block	
	DIR	Direct: Values are changed via direct assignment and take effect at the start of the next MC_Servo.
	CAL	At call of Motion Control instruction: Values are changed directly and take effect at the start of the next MC_Servo after the call of the corresponding Motion Control instruction in the user program.
	RES	Restart: Changes to the start value in the load memory are made using the extended instruction "WRIT_DBL" (write to DB in load memory). Changes will not take effect until after restart of the technology object.
	RON	Read only: The tag cannot and must not be changed during runtime of the user program.
Description	Description of the tag	

Access to the tags is with "<TO>.<tag name>". The placeholder <TO> represents the name of the technology object.

11.2 Tags of the interpreter technology object (S7-1500T)

11.2.1 Interpreter program and Interpreter mapping (Interpreter) (S7-1500T)

The following variables display information about the Interpreter program and Interpreter mapping.

Variables

Legend (Page 361)

Variable	Data type	Values	W	Description	
ProgramName	STRING	-	RON	Name of the currently loaded technology object Interpreter program	
MappingName	STRING	-	RON	Name of the currently loaded Interpreter mapping technology object	
ProgramSource	DINT	0 ... 2	RON	Source of the currently loaded Interpreter program	
				0	No Interpreter program loaded
				1	Interpreter program technology object
				2	Reserved
MappingSource	DINT	0 ... 3	RON	Source of the currently loaded Interpreter mapping	
				0	No Interpreter mapping loaded
				1	Interpreter mapping technology object
				2	Reserved
				3	
ActualLineNumber	UDINT	0 ... 4294967295	RON	Line number of the currently executed program line or the last executed program line	

11.2.2 "Parameter" variable (Interpreter) (S7-1500T)

The tag structure "<TO>.Parameter.<Tag name>" contains parameters for preparation of the Interpreter program.

Tags

Legend (Page 361)

Tag	Data type	Values	W	Description
Parameter	TO_Struct_Interpreter_Parameter			
MaxNumberOfCommands	DINT	10 ... 100	RON	Maximum number of jobs to be prepared in the interpreter job sequence
StartTimeout	LREAL	0.0 ... 2.0	DIR	Maximum wait time Maximum time between the possible and effective start of an order in [s] An incompletely prepared motion sequence will start after the expiry of the maximum wait time. 0.0: Motion sequences only start when they are fully prepared or the maximum number of jobs to be prepared in the interpreter job sequence has been reached.
ProgramOverride	LREAL	1.0 ... 100.0	CAL	Start value of the program override in [%] The percentage factor acts as the start value of the modal parameter when preparing the dynamic profiles of axis and kinematics motions.

11.2.3 "Clipboard" variable (Interpreter) (S7-1500T)

The variable structure "<TO>.Clipboard.<variable name>" contains the clipboard variables of the Interpreter technology object.

Variables

Legend (Page 361)

Variable	Data type	Values	W	Description
Clipboard	TO_Struct_Interpreter_Clipboard			Clipboard variables
CbBool	ARRAY [1..300] of BOOL	-	DIR	Area for data exchange with the Interpreter program for BOOL tags
CbDint	ARRAY [1..100] of DINT	-2147483648 ... 2147483647	DIR	Area for data exchange with the Interpreter program for DINT tags
CbLreal	ARRAY [1..100] of LREAL	-1.7977e+308 .. 1.7977e+308	DIR	Area for data exchange with the Interpreter program for LREAL tags

11.2.4 "StatusInterpreter" variable (Interpreter) (S7-1500T)

The variable structure "<TO>.StatusInterpreter.<variable name>" contains status information of the technology object.

Variables

Legend (Page 361)

Variable	Data type	Values	W	Description
StatusInterpreter	TO_Struct_Interpreter_StatusInterpreter			
ProgramMode	DINT	0 ... 3	RON	Operating mode of Interpreter program execution
				0 Automatic
				2 Debug
AssignedKinematics	DB_ANY	0 to 65535	RON	Technology object data block of the assigned kinematics technology object
LevelOfPreparedCommands	LREAL	0.0 to 100.0	RON	Capacity utilization of the Interpreter job sequence in [%] in 5% increments

11.2.5 "StatusWord" variable (Interpreter) (S7-1500T)

The "<TO>.StatusWord" tag contains the status information of the Interpreter technology object.

Information on evaluating individual bits (e.g. bit 2 "RestartActive") can be found in the "Evaluating StatusWord, ErrorWord, and WarningWord" section of the "S7-1500/S7-1500T Motion Control overview" documentation (Page 10).

Tags

Legend (Page 361)

Tag	Data type	Values	W	Description
StatusWord	DWORD	-	RON	
Bit 0	-	-	-	"Control" A Motion Control job is active at the Interpreter technology object, or the toolbar of the programming editor has the master control.
Bit 1	-	-	-	"Error" An error is present.
Bit 2	-	-	-	"RestartActive" A restart is active. The technology object will be reinitialized.
Bit 3	-	-	-	"OnlineStartValuesChanged" The restart tags have been changed. For the changes to be applied, the technology object must be reinitialized.
Bit 4	-	-	-	"MasterControlActive" The toolbar of the programming editor has master control over the Interpreter technology object.

Tag	Data type	Values	W	Description
Bit 5	-	-	-	"InRun" The technology object is executing an Interpreter program.
Bit 6	-	-	-	"Done" Execution of the Interpreter program has finished.
Bit 7	-	-	-	"Stopping" Execution of the Interpreter program will be or has been stopped.
Bit 8	-	-	-	"Interrupted" Execution of the Interpreter program has been interrupted.
Bit 9	-	-	-	"Loading" The technology object is loading the Interpreter program. Interpreter program preparation is running.
Bit 10	-	-	-	"Loaded" The Interpreter program is loaded and prepared.
Bit 11 ... Bit 31	-	-	-	Reserved

11.2.6 "ErrorWord" tag (interpreter) (S7-1500T)

The "<TO>.ErrorWord" tag indicates technology object errors (technology alarms).

Information on evaluating individual bits (e.g. bit 3 "CommandNotAccepted") can be found in the "Evaluating StatusWord, ErrorWord, and WarningWord" section of the "S7-1500/S7-1500T Motion Control overview" documentation (Page 10).

Tags

Legend (Page 361)

Tag	Data type	Values	W	Description
ErrorWord	DWORD	-	RON	
Bit 0	-	-	-	"SystemFault" A system-internal error has occurred.
Bit 1	-	-	-	"ConfigFault" Configuration error One or more configuration parameters are inconsistent or invalid.
Bit 2	-	-	-	"UserFault" Error in user program at a Motion Control instruction or its use
Bit 3	-	-	-	"CommandNotAccepted" Job cannot be executed A Motion Control instruction cannot be executed because the necessary conditions have not been met.

Tag	Data type	Values	W	Description
Bit 4	-	-	-	"UserProgramFault" Error in the current interpreter program
Bit 5	-	-	-	"UserMappingFault" Error in the current interpreter mapping
Bit 6 ... Bit 31	-	-	-	Reserved

11.2.7 "ErrorDetail" tag (interpreter) (S7-1500T)

The tag structure "<TO>.ErrorDetail.<tag name>" contains the alarm number and the effective local alarm response for the technology alarm that is currently pending on the technology object.

You can find a list of the technology alarms and alarm responses in the "Technology alarms overview" section of the "S7-1500/S7-1500T Motion Control alarms and error IDs" documentation (Page 10).

Tags

Legend (Page 361)

Tag	Data type	Values	W	Description	
ErrorDetail	TO_Struct_Interpreter_ErrorDetail				
Number	UDINT	-	RON	Alarm number	
Reaction	DINT	0, 14, 15	RON	Effective alarm response	
				0	No reaction (warnings only)
				14	Stop execution of the interpreter program with diagnostics option
				15	Stop execution of the interpreter program
LineNumber	UDINT	0 ... 4294967295	RON	Number of the faulty line of the interpreter program or interpreter mapping	
ErrorInfo	STRING	-	RON	Additional error information	

11.2.8 "WarningWord" tag (interpreter) (S7-1500T)

The "<TO>.WarningWord" tag indicates pending warnings at the technology object.

Information on evaluating individual bits (e.g. bit 2 "UserWarning") can be found in the "Evaluating StatusWord, ErrorWord, and WarningWord" section of the "S7-1500/S7-1500T Motion Control overview" documentation (Page 10).

Tags

Legend (Page 361)

Tag	Data type	Values	W	Description
WarningWord	DWORD	-	RON	
Bit 0	-	-	-	"SystemWarning" A system-internal error has occurred.
Bit 1	-	-	-	"ConfigWarning" Configuration error One or more configuration parameters are inconsistent or invalid.
Bit 2	-	-	-	"UserWarning" Error in user program at a Motion Control instruction or its use
Bit 3	-	-	-	"CommandNotAccepted" Job cannot be executed A Motion Control instruction cannot be executed because the necessary conditions have not been met.
Bit 4	-	-	-	"UserProgramWarning" Error in the current interpreter program
Bit 5	-	-	-	"UserMappingWarning" Error in the current interpreter mapping
Bit 6 ... Bit 31	-	-	-	Reserved

11.2.9 "ControlPanel" tag (Interpreter) (S7-1500T)

The tag structure "<TO>.ControlPanel.<tag name>" contains no relevant data for you. This tag structure is internally used.

11.2.10 "InternalToTrace" tag (Interpreter) (S7-1500T)

The tag structure "<TO>.InternalToTrace.<tag name>" contains no relevant data for you. This tag structure is internally used.

Glossary

Breakpoint

A breakpoint is a deliberately set marking at a point in the MCL program. Program execution in "Debug" mode is interrupted at a breakpoint and can be resumed.

Debug program mode

In "Debug" program mode, the program execution by the Interpreter can be influenced via the user program of the CPU and the toolbar in the programming editor.

Interpreter

The Interpreter executes a series of serial instructions that are stored in an Interpreter program.

Interpreter job sequence

The Interpreter job sequence contains the jobs of the Interpreter program. The length of the Interpreter job sequence specifies the maximum number of jobs that can be prepared by the Interpreter.

Interpreter mapping

An Interpreter mapping defines an assignment of objects in the Interpreter program to objects in the user program.

Interpreter program

The Interpreter program contains the motion jobs for the kinematics and/or single axes.

MCL

Motion Control Language (MCL) is an interpretive programming language for specifying motion jobs and other technology functions.

The language is based on the Structured Text (ST) programming language specified in DIN EN-61131-3 (IEC 61131-3).

MCL instruction

An MCL instruction is an individual command within an MCL program that is performed sequentially by the Interpreter. It is used to control movements, perform calculations or execute additional technology functions.

Motion sequence

A motion sequence refers to several motion jobs that are blended into one another. A motion job that is not blended with either the previous or the subsequent motion job is an independent motion sequence in its own right.

Program execution

During program execution, the already prepared jobs are executed by the interpreter technology object or the technology objects controlled by the interpreter.

Program override

Percentage correction of the dynamic values for path jobs in the Interpreter program. Affects velocity, acceleration and deceleration in the program execution of the path jobs. Takes effect in addition to a velocity override at the kinematics or at an individual kinematics axis.

Program preparation

The Interpreter technology object interprets the jobs from the loaded Interpreter program by adding the jobs to the Interpreter sequence and preparing them. When the interpreter program is loaded and during program execution, the interpreter prepares the jobs in advance.

Synchronized action

A synchronized action is a job that runs in parallel with other jobs, such as opening a gripper during a kinematics motion.

Index

T

V