

BECKHOFF New Automation Technology

Manual | EN

TF5100

TwinCAT 3 | NC I

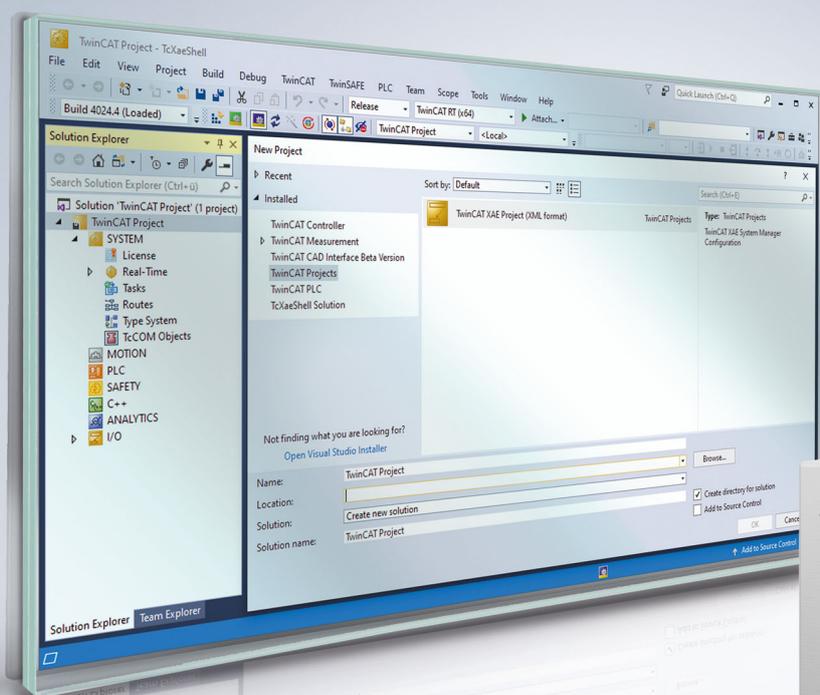


Table of contents

1	Foreword	7
1.1	Notes on the documentation	7
1.2	Safety instructions	8
1.3	Notes on information security.....	9
2	Introduction	10
3	User interface in the TwinCAT 3 Engineering environment	11
3.1	Outline	11
3.2	Interpolation Channel	12
3.3	Interpreter element.....	14
3.3.1	Interpreter online window	15
3.3.2	"Interpreter" tab	17
3.3.3	"M-Functions" tab.....	18
3.3.4	"R parameters" tab.....	19
3.3.5	"Zero point" tab	20
3.3.6	"Tools" tab.....	21
3.3.7	"Editor" tab	22
3.3.8	"MDI" tab	23
3.4	Group element	23
3.4.1	"General" tab.....	24
3.4.2	"DXD" tab	24
3.4.3	"Settings" tab.....	28
3.4.4	"Online" tab	29
3.4.5	"3D-Online" tab	30
4	GST Reference Manual	31
4.1	General Notes	31
4.2	Preprocessor	31
4.3	Combining G-Code and ST.....	33
4.4	G-Code (DIN 66025)	35
4.4.1	Comments.....	35
4.4.2	Execution Order	36
4.4.3	Mutual Exclusive G-Codes.....	37
4.4.4	Rapid Traverse (G00)	37
4.4.5	Linear Interpolation (G01)	38
4.4.6	Circular Interpolation (G02, G03, IJK, U)	38
4.4.7	Dwell Time (G04)	40
4.4.8	Accurate Stop (G09,G60).....	40
4.4.9	Delete Distance to go (G31).....	40
4.4.10	Zero Offset Shifts (G53,G54...59)	41
4.4.11	Tool Radius Compensation (D, G40, G41, G42)	42
4.4.12	Working Plane and Feed Direction (G17, G18, G19, P)	44
4.4.13	Inch/metric dimensions (G70, G71, G700, G710).....	45
4.4.14	Dimensional Notation (G90, G91).....	47
4.4.15	M-Functions (M).....	47

4.4.16	General Codes (F, N, Q, X, Y, Z, A, B, C).....	48
4.5	ST - Structured Text (IEC 61131-3)	51
4.5.1	Comments.....	51
4.5.2	Literals.....	51
4.5.3	Native Data Types.....	54
4.5.4	Userdefined Types	55
4.5.5	Control Structures	56
4.5.6	Userdefined Functions	57
4.5.7	Standard Functions	59
4.5.8	R-Parameters.....	66
4.6	CNC Functions.....	67
4.6.1	Strings and Messages.....	67
4.6.2	Transformations	68
4.6.3	Circular Movement.....	75
4.6.4	Centerpoint Correction.....	75
4.6.5	Tools	76
4.6.6	Synchronization.....	78
4.6.7	Query of Axes	78
4.6.8	Current Point.....	79
4.6.9	Tool Radius Compensation.....	80
4.6.10	Suppression of G-Code Blocks	81
4.6.11	Zero Offset Shift.....	82
4.6.12	Units.....	83
4.6.13	Trigonometric (Unit Aware)	84
4.6.14	Feed Mode	85
4.6.15	Feed Interpolation	85
4.6.16	Streaming of Large G-Code Files	86
4.6.17	Vertex Smoothing.....	86
4.6.18	Automatic Accurate Stop.....	88
4.6.19	Spline Interpolation	89
4.6.20	Dynamic Override	92
4.6.21	Center Point Reference of Circles.....	93
4.6.22	Change in axis dynamics	93
4.6.23	Change in path dynamics.....	94
4.7	Transformations	95
4.7.1	Modification of the Effective Transformation T and its Effect.....	95
4.7.2	Components of the Effective Transformation T.....	96
4.7.3	Applying Transformations	96
4.7.4	Revoking Transformations	97
4.7.5	Restoration of Stack.....	97
4.8	Error Reporting.....	98
4.8.1	Error Messages.....	98
4.8.2	Compile-Time Errors and Runtime Errors.....	98
4.8.3	Errors in G-Code	99
4.8.4	Preprocessing	100
4.9	General Command Overview	101

4.10	Comparative Command Overview	110
5	Classic Dialect Reference Manual	122
5.1	Basic Principles of NC Programming	122
5.1.1	Structure of an NC Program.....	122
5.1.2	Block Skipping.....	123
5.1.3	Look-Ahead.....	123
5.1.4	Smoothing of Segment Transitions	125
5.1.5	Co-ordinate System	125
5.1.6	Dimensional Notation	126
5.1.7	Working Plane and Feed Direction	126
5.1.8	Inch/metric dimensions	128
5.1.9	Single Block Operation.....	129
5.1.10	Arithmetic Parameters.....	130
5.2	Programming Movement Statements.....	133
5.2.1	Referencing.....	133
5.2.2	Rapid Traverse.....	133
5.2.3	Linear Interpolation	134
5.2.4	Circular Interpolation	135
5.2.5	Helix	137
5.2.6	Dwell Time	137
5.2.7	Accurate Stop.....	138
5.2.8	Feed interpolation	138
5.2.9	Zero Offset Shifts	139
5.2.10	Target Position Monitoring	142
5.2.11	Contour definitions	144
5.2.12	Rotation.....	145
5.2.13	Mirror.....	148
5.2.14	Smoothing of segment transitions.....	149
5.2.15	Circular Smoothing.....	154
5.2.16	Automatic Accurate Stop.....	155
5.2.17	Delete Distance to Go	156
5.2.18	Modulo Movements.....	156
5.2.19	Auxiliary axes	158
5.3	Supplementary Functions	161
5.3.1	M-Functions	161
5.3.2	H, T and S Parameters	165
5.3.3	Decoder stop.....	166
5.3.4	Jumps.....	167
5.3.5	Loops	169
5.3.6	Subroutine techniques	170
5.3.7	Dynamic Override	172
5.3.8	Altering the Motion Dynamics	172
5.3.9	Change of the Reduction Parameters.....	173
5.3.10	Change of the Minimum Velocity	175
5.3.11	Read Actual Axis Value.....	176
5.3.12	Skip virtual movements	177

5.3.13	Messages from NC program	177
5.4	Tool Compensation	177
5.4.1	Tool Data	177
5.4.2	Selecting and Deselecting the Length Compensation	180
5.4.3	Cartesian Tool Translation	180
5.4.4	Cutter Radius Compensation	183
5.4.5	Orthogonal Contour Approach/Departure	188
5.4.6	Path Velocity in Arcs	188
5.4.7	Bottle Neck Detection	189
5.5	Command overview	190
5.5.1	General command overview	190
5.5.2	@-Command Overview	193
6	PLC NCI Libraries	195
6.1	PLC Library: Tc2_NCI	195
6.1.1	Configuration	195
6.1.2	NCI POU's	202
6.1.3	Parts program generator	260
6.1.4	Blocks for compatibility with existing programs	267
6.1.5	Obsolete	293
6.2	PLC Library: Tc2_PlcInterpolation	294
6.2.1	FB_NciFeedTablePreparation	296
6.2.2	FB_NciFeedTable	297
6.2.3	Types and Enums	298
7	Samples	311
8	Appendix	312
8.1	Display of the parts program	312
8.2	Display of technology data	314
8.3	Displaying the remaining path length	318
8.4	Parameterisation	319
8.4.1	Path override (interpreter override types)	322
8.5	Cyclic Channel Interface	323

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.

It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.

It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without prior announcement. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.



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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.



Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our <https://www.beckhoff.com/secguide>.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at <https://www.beckhoff.com/secinfo>.

2 Introduction

The TwinCAT NCI stands for 'numerical control interpolation' and is the NC system for interpolated path movements.

TwinCAT NCI offers 3D interpolation (interpreter, setpoint generation, position controller), an integrated PLC with an NC interface and an I/O connection for axes via the fieldbus.

NCI can be used to drive 3 path axes and up to 5 auxiliary axes per channel. In addition, master/slave couplings can be formed. In combination with TwinCAT Kinematic Transformation (TF511x), complex kinematic systems can be controlled via NCI.

Programming is done with a dedicated NC program, based on DIN 66025, with its own language extensions (cf. [Classic Dialect Reference Manual \[▶ 122\]](#)) or directly from the PLC with the [PLC Library: Tc2_PlcInterpolation \[▶ 294\]](#).

Installation preconditions

TwinCAT NCI is integrated in the TwinCAT 3 installation.

Target system

Windows 7, Windows 10, Windows CE (only Classic Interpreter)

Minimum Plattform-Level: 40

Overview

Chapter	Contents
XAE user interface [▶ 11]	Description of the parameters and functionalities for the interpreter in the TwinCAT 3 Engineering environment (XAE)
Interpreter [▶ 122]	Interpreter programming instructions.
PLC NCI Libraries [▶ 195]	Description of the special NCI libraries
Samples [▶ 311]	Samples for using TwinCAT NCI with PLC and parts program, and for direct motion control from the PLC with the Tc2_PlcInterpolation library
Appendix [▶ 319]	Parameterization, cyclic channel interface

Further information

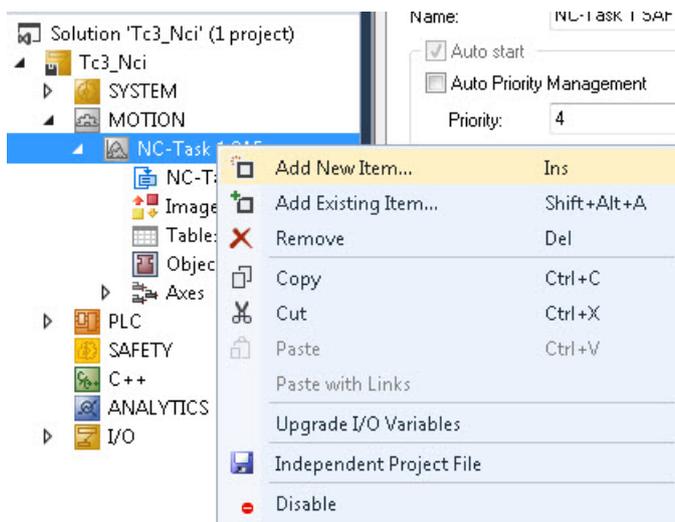
- [ADS Return Codes](#)
- [ADS Specification of the NC](#)

3 User interface in the TwinCAT 3 Engineering environment

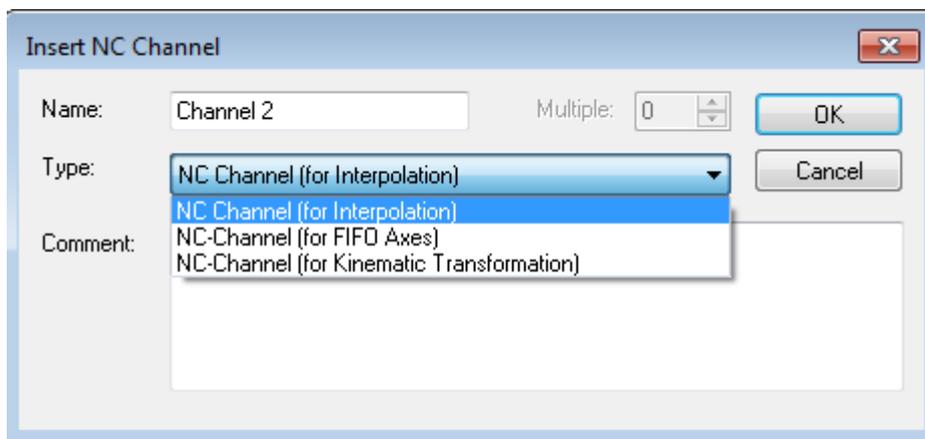
3.1 Outline

In order to be able to use the interpolation, add an interpolation channel in the XAE . This applies to the interpreter and the [PLC Library: Tc2_PlcInterpolation \[► 294\]](#).

1. Create an NC channel.



2. In the selection box select the NC channel for the interpolation.



3. Assign PTP axes to it from the PLC via a function block.

⇒ The created channel consists of the following elements:

[Interpolation Channel \[► 12\]](#)

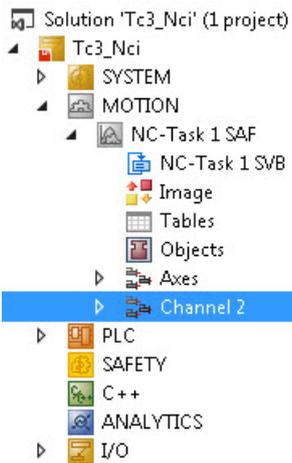
Description of the properties pages embedded in the 'interpolation' element.

[Interpreter Element \[► 14\]](#)

Description of the properties pages embedded in the 'Interpreter' element

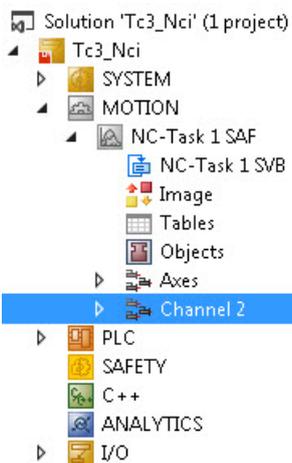
[Group element \[► 23\]](#)

Description of the properties pages embedded in the 'group' element



Note Axis-specific parameters for NCI can be found in the axis parameterization under subitem 'NCI parameters'.

3.2 Interpolation Channel

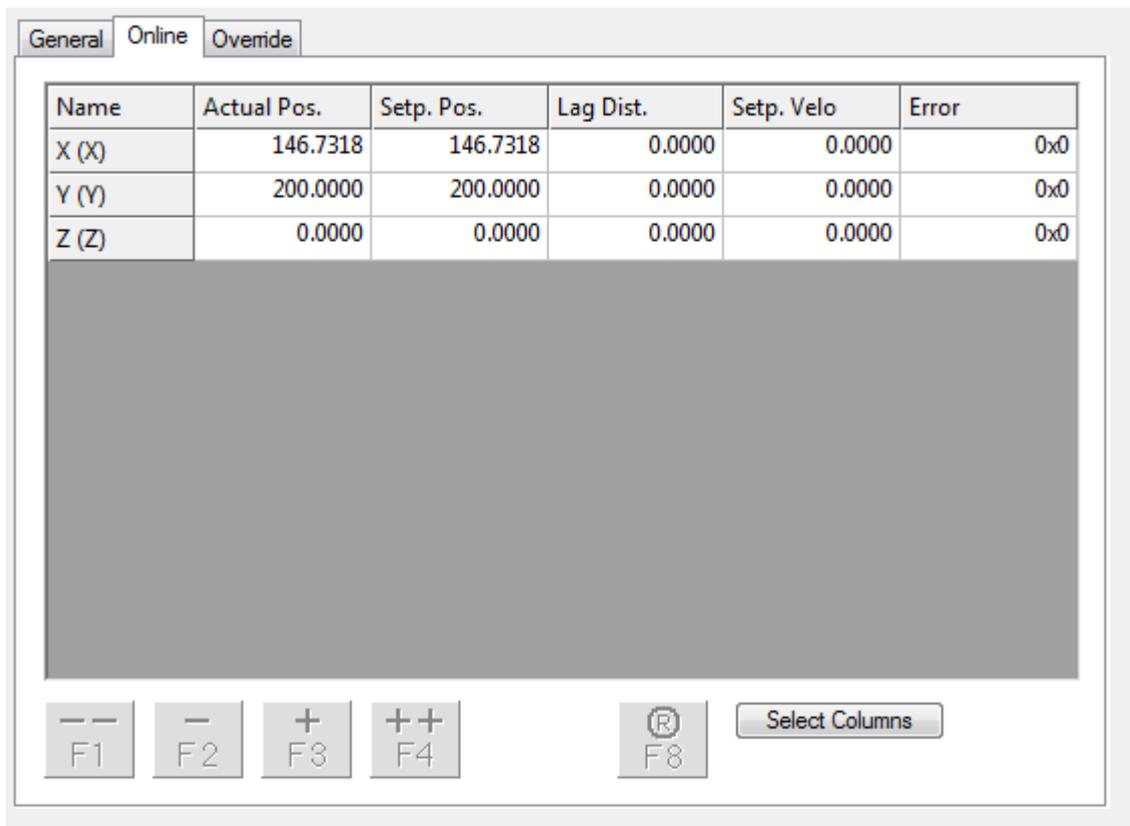


Click on the interpolation channel to display the following dialogs:

"Online" tab

All the axes in the current Interpolation Group [▶ 23] will be listed. Currently shown:

- Actual positions
- Set positions
- Following errors
- Set velocities and
- Error Codes



Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	Error
X (X)	146.7318	146.7318	0.0000	0.0000	0x0
Y (Y)	200.0000	200.0000	0.0000	0.0000	0x0
Z (Z)	0.0000	0.0000	0.0000	0.0000	0x0

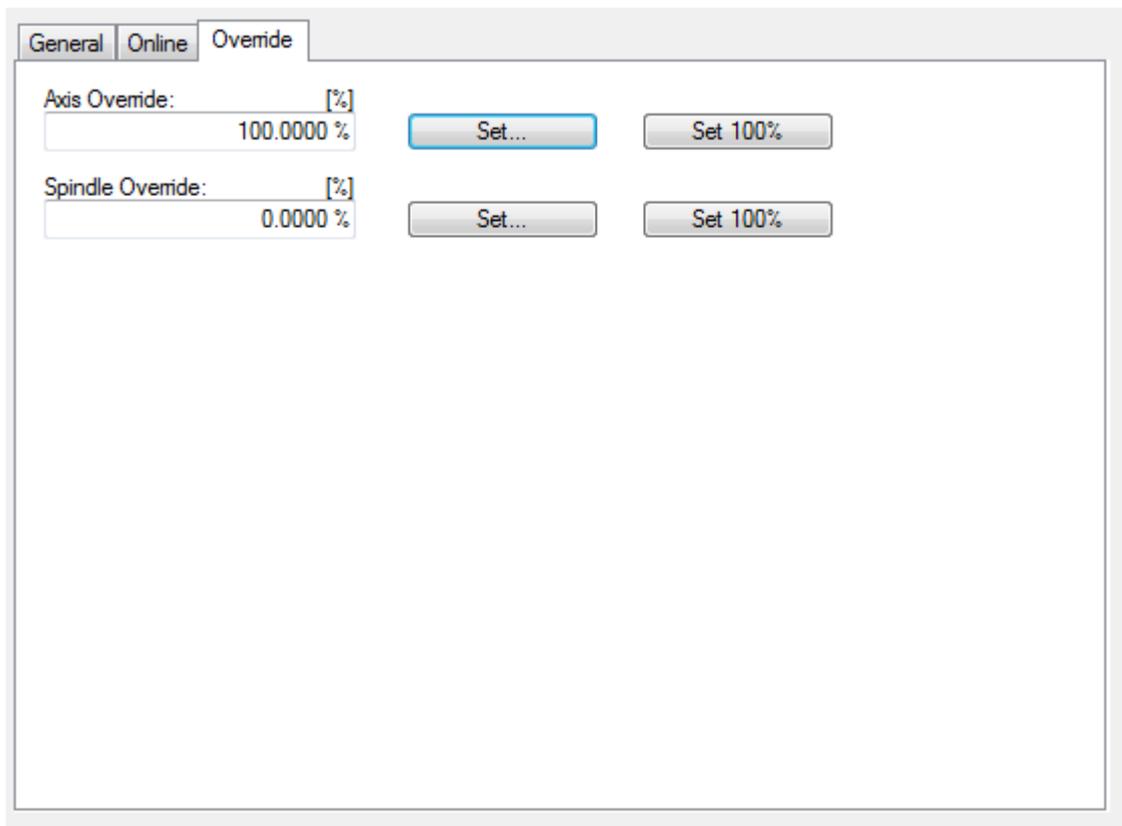
Below the table, there are several function keys: F1 (two minus signs), F2 (one minus sign), F3 (one plus sign), F4 (two plus signs), F8 (a registered trademark symbol), and a 'Select Columns' button.

"Override" tab

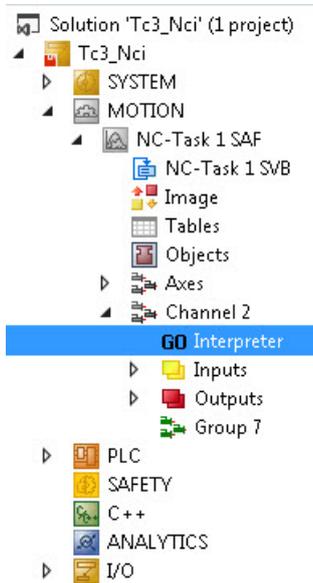
The channel override for the axes can be read and set on the 'Override' page. If PLC is running and the [cyclical channel interface](#) [► 323] is being written, the override set here will be overwritten by the PLC.

Further information on the override principle can be found under [Path override \(interpreter override types\)](#) [► 322].

The spindle override is described by the cyclic channel interface, although it is currently not supported.



3.3 Interpreter element



Click on "Interpreter" to show the following property pages and the online window:

3.3.1 Interpreter online window

Name	Actual Pos.	Setp. Pos.	Lag Dist.	Setp. Velo	Er...
X (X)	3207.7262	3207.7262	0.0000	99.9180	0x0
Y (Y)	1988.3170	1988.3170	0.0000	3.9967	0x0
Z (Z)	0.0000	0.0000	0.0000	0.0000	0x0
Q1 (Q1)	0.0000	0.0000	0.0000	0.0000	0x0
Q2 (Q2)	0.0000	0.0000	0.0000	0.0000	0x0

Actual Program Line:

```
N20 G01 X1000
N30 G01 X3000
N40 G01 X3500 Y2000
```

Program Name:

Interpreter State: Buffer Size (Byte):

Channel State:

Axes

As on the "Online" properties page in the interpolation channel, this window lists all axes currently included in the interpolation group. Values for the following parameters are displayed:

- Actual positions
- Set positions
- Following errors
- Set velocities and
- Current error codes

Actual Program Line

The Actual Program Line shows the current NC block to be processed in the block execution. The last row in the window is the current block.

Unlike this, the current block is in the middle row in the case of GST.

As for nearly all the parameters, the program display can be read off via ADS. This can be used to display the current NC blocks in a Visual Basic application, for example (see ADS device documentation - ADS Interface NC).

Program name

Displays the name of the currently loaded program. This does not necessarily have to be the program displayed in Editor.

Interpreter status

The interpreter status indicates the current status of the interpreter state machine. The complete list is given below. As PLC evaluation does not require all status information, only the most important parameters are explained.

Status	Description
ITP_STATE_IDLE	The interpreter is in idle state when there is no NC program loaded as yet or when a group reset is being executed. The interpreter also goes into idle state

Status	Description
	when a current program is stopped. In the case a group reset must be executed in order to prevent error 0x42C5. It is therefore recommended to execute a group reset after stopping via the PLC.
ITP_STATE_READY	After successful loading of an NC program, the interpreter is in ready state. After a program has been successfully processed and exited, the interpreter goes into ready state. In the meantime, however, other states are accepted.
ITP_STATE_ABORTED	If a runtime error occurs during the processing of an NC program, the interpreter goes into aborted state. The actual error code is given in the channel status.
ITP_STATE_SINGLESTOP	This status is only accepted in <u>Single Block Mode</u> [▶ 129]. As soon as the entry has been sent from the interpreter to the NC core, the interpreter goes into this mode.

● Querying the interpreter status during program execution

i Since the interpreter status may change between different states during program execution, we recommend querying it with a negative logic. During program execution the interpreter state is not necessarily ITP_STATE_RUNNING. If the program was executed successfully, the interpreter is subsequently always in Ready state (see also [Samples](#) [[▶ 311](#)]).

● End of program

i The end of the program is characterized by an M function. Therefore either M2 or M30 are being used. If the M function is missing at the end of the program, the status of the interpreter could return wrong values.

Interpreter status return values

```

0 ITP_STATE_INITFAILED
1 ITP_STATE_IDLE
2 ITP_STATE_READY
3 ITP_STATE_STARTED
4 ITP_STATE_SCANNING
5 ITP_STATE_RUNNING
6 ITP_STATE_STAY_RUNNING
7 ITP_STATE_WRITETABLE
8 ITP_STATE_SEARCHLINE
9 ITP_STATE_END
10 ITP_STATE_SINGLESTOP
11 ITP_STATE_ABORTING
12 ITP_STATE_ABORTED
13 ITP_STATE_FAULT
14 ITP_STATE_RESET
15 ITP_STATE_STOP
16 ITP_STATE_WAITFUNC
17 ITP_STATE_FLUSHBUFFERS

```

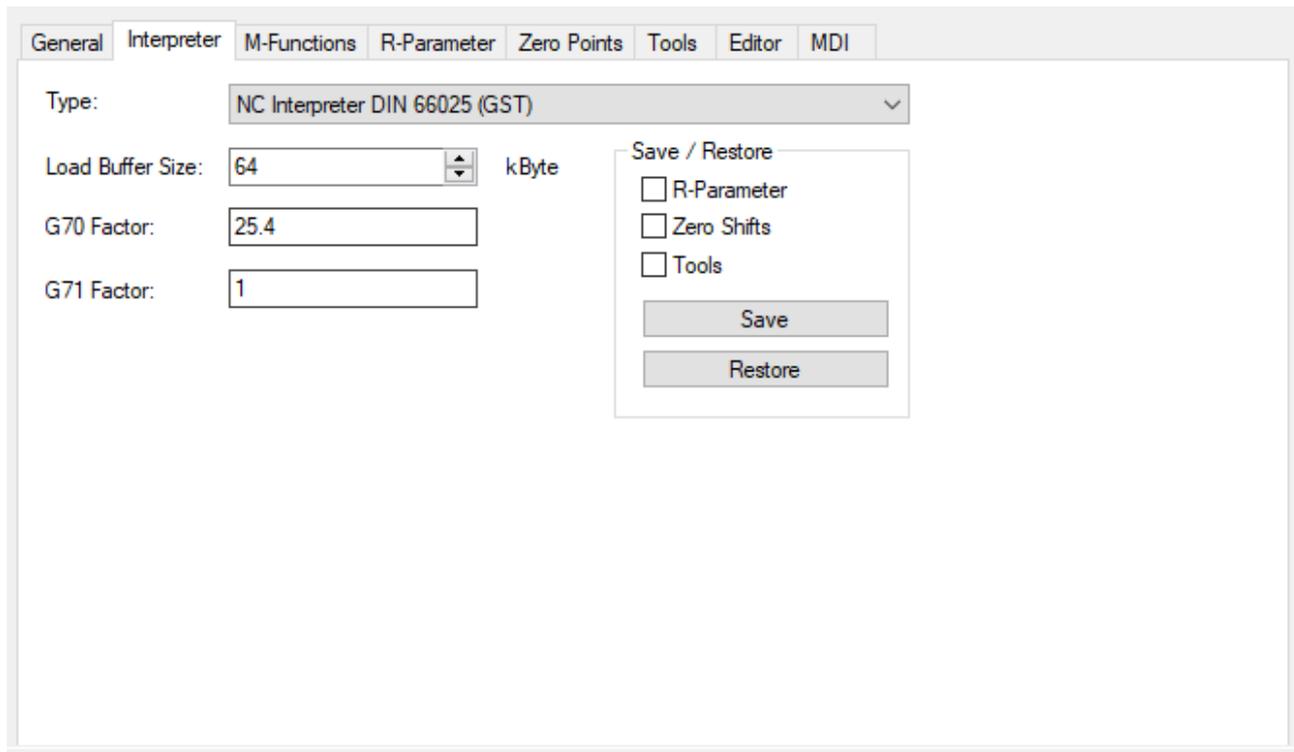
Channel status

The channel status indicates the current error state of the channel. If an error occurs during NC program loading or runtime, the corresponding error code is displayed here. If, for example, an axis following error occurs during processing, the NC program is stopped and the channel status will have a value unequal 0. The channel status should therefore always be checked in the PLC, in order to be able to respond to errors. The channel status is always 0 during normal operation.

Loading buffer

The current size of the loading buffer for the interpreter is displayed here. Select the "Interpreter" tab to change the value.

3.3.2 "Interpreter" tab



Type

The interpreter type can be selected in the Type selection box. Available are

- the [GST-interpreter \[► 31\]](#). GST combines native DIN 66025 based G-code with programming extensions of Structured Text as a higher level language.
- The DIN 66025 based [NC-interpreter \[► 122\]](#) (Classic Dialect) with @-command register function extensions.
- The selection of none if the [PlcInterpolation \[► 294\]](#) library is used.

As default setting the GST-interpreter is set. To employ the NC-interpreter with register function extensions you have to select it explicitly.

Loading Buffer Size

The loading buffer for the interpreter can be edited here. Note that the memory required in the interpreter is substantially greater than the size of the NC-file. The maximum permitted loading buffer size is limited to 64 MB.

● Changing the Loading Buffer Size

i If the size of the loading buffer is changed, it is absolutely necessary to execute a TwinCAT restart.

G70/G71 Factor

If a switch from [G71 \[► 128\]](#) (millimeters - default) to G70 takes place in the parts program, the conversion factor is stored here. This conversion factor only has to be edited if the base reference system is not millimeters.

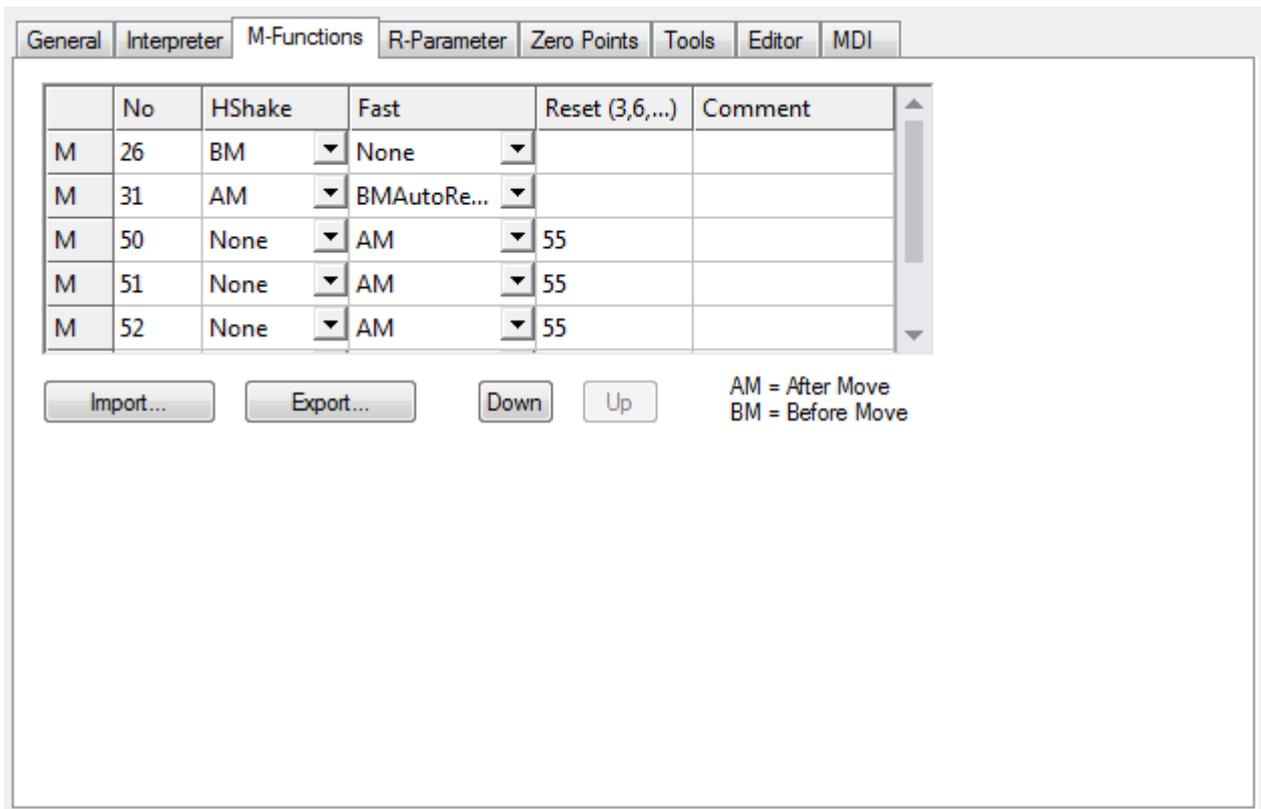
If for example the machine was calibrated based on inches and G70 is activated in the parts program, the G70 factor should be set to 1 and the G71 factor should be set to 1/25.4.

Save/Restore

At runtime the Save function can be used to save a “snapshot” of the current parameters. The checkboxes can be used to specify the parameters to be saved. The Save function generates the file ‘SnapShot.bin’ in the TwinCAT\CNC directory.

The Restore function loads the file saved with the Save function. This function is solely intended for debugging purposes.

3.3.3 "M-Functions" tab



● Use only with interpreter

i This tab is irrelevant for operation with the library Tc2_PlcInterpolation.

Shows the currently parameterized M-functions. On this page new M-functions can be added, or existing ones modified.

A more detailed description of the available parameters can be found in the interpreter description under [M-functions \[► 161\]](#).

● Parameterization of M-functions

i If M-functions are re-parameterized, subsequent activation of the configuration and a TwinCAT restart is required.

3.3.4 "R parameters" tab

Parameter Range	Value 1	Value 2	Value 3	Value 4	Value 5
R 0- 4	9.000000	789.000000	0.000000	56.000000	0.000000
R 5- 9	0.100000	0.005000	45.000000	0.000000	45.000000
R 10- 14	0.000000	45.000000	0.000000	7.100000	0.000000
R 15- 19	78.000000	0.000000	456.000000	0.000000	0.000000
R 20- 24	0.000000	0.000000	0.000000	0.000000	0.000000
R 25- 29	80.000000	120.000000	5846.000000	0.000000	0.000000
R 30- 34	0.000000	0.000000	0.000000	0.188900	0.000000
R 35- 39	0.000000	0.000000	0.000000	0.000000	0.000000

The currently applicable R parameters are displayed on the 'R parameters' properties page. During the test phase it is possible to, for example, initialize or change R parameters here. R parameters are generally edited, however, from the NC program or if necessary, from the PLC.

You can find further information about R parameters in the interpreter description under [R Parameters](#) [► 130].

3.3.5 "Zero point" tab

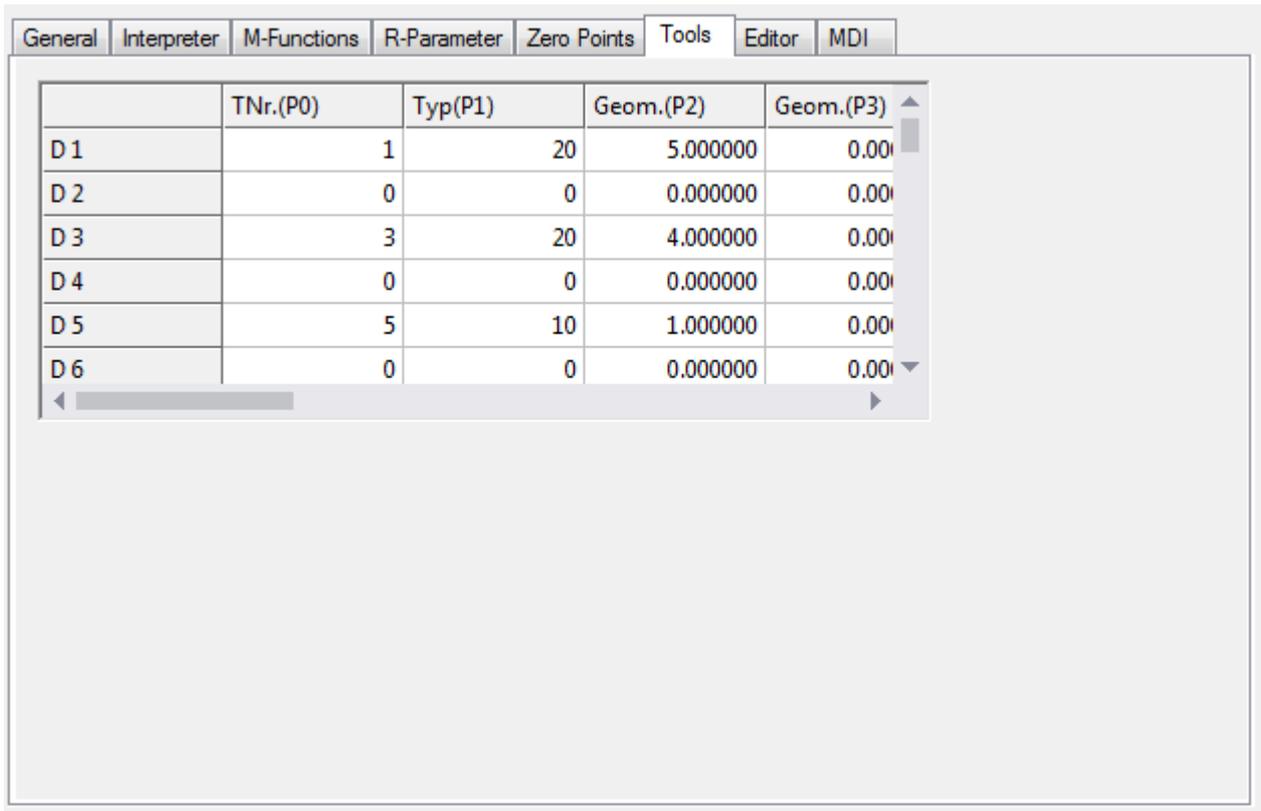
	P54 F	P54 G	P55 F	P55 G	P56 F	P56 G	P57 F	P57
X	100.000...	50.0000...	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
Y	10.0000...	20.0000...	1.000000	0.000000	0.000000	0.000000	0.000000	0.0
Z	45.0000...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0

The current zero shift values for the axes within the interpolation group are displayed here. The parameters P54..P59 represent for the corresponding G code. As for the R parameters, the zero shift values can be edited from here.

Note Columns F & G (e.g. P54 F & P54 G) exist for historical reasons and are added for each parameter.

You can find further details of the effects in the interpreter description under [zero shifts \[►_139\]](#).

3.3.6 "Tools" tab

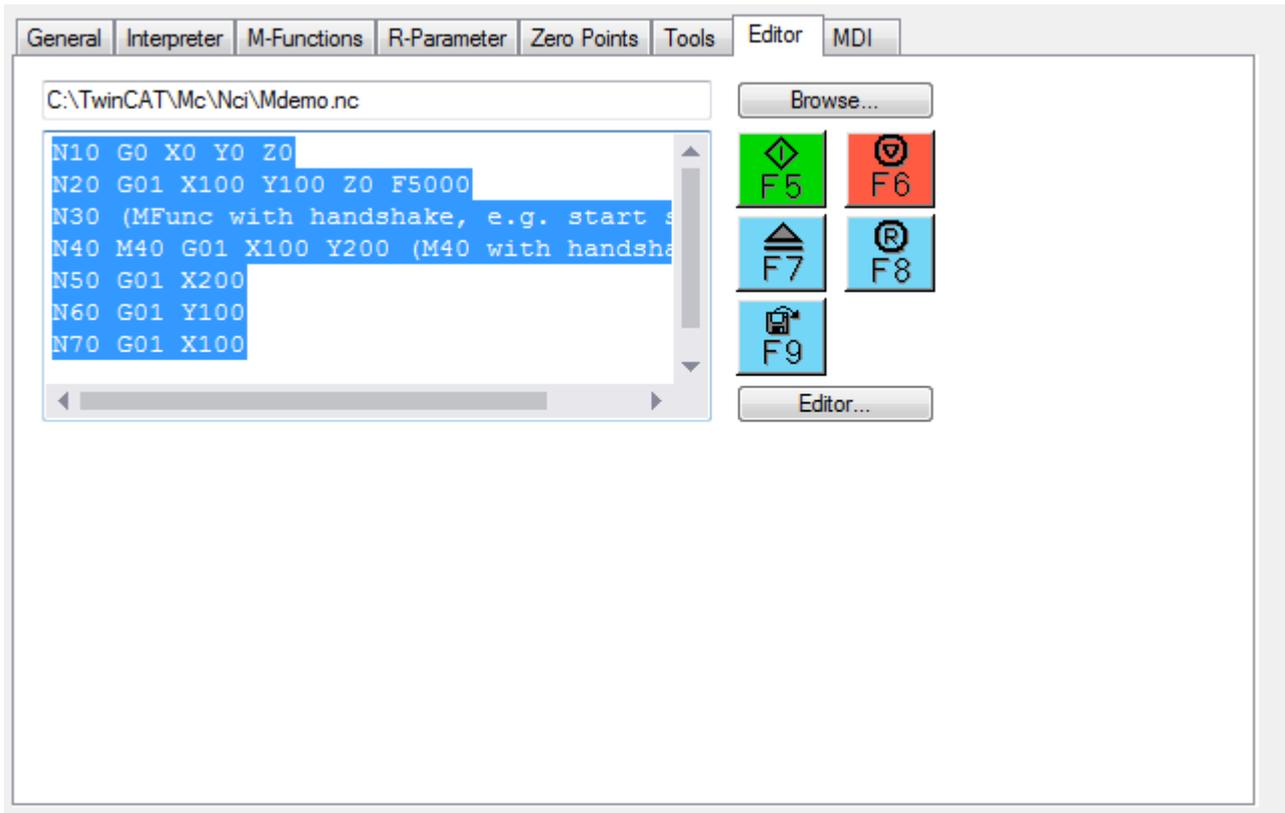


	TNr.(P0)	Typ(P1)	Geom.(P2)	Geom.(P3)
D 1	1	20	5.000000	0.00
D 2	0	0	0.000000	0.00
D 3	3	20	4.000000	0.00
D 4	0	0	0.000000	0.00
D 5	5	10	1.000000	0.00
D 6	0	0	0.000000	0.00

You can edit the data for the tool compensation on the "Tools" property page.

More detailed parameter descriptions can be found in the interpreter description under [tool compensations](#) [[▶ 177](#)].

3.3.7 "Editor" tab



The editor is used to display and edit the NC programs.

- **Browse...**
Opens a dialog with which existing NC programs can be selected and displayed.

i Remote Connection: Load NC-File from Target System

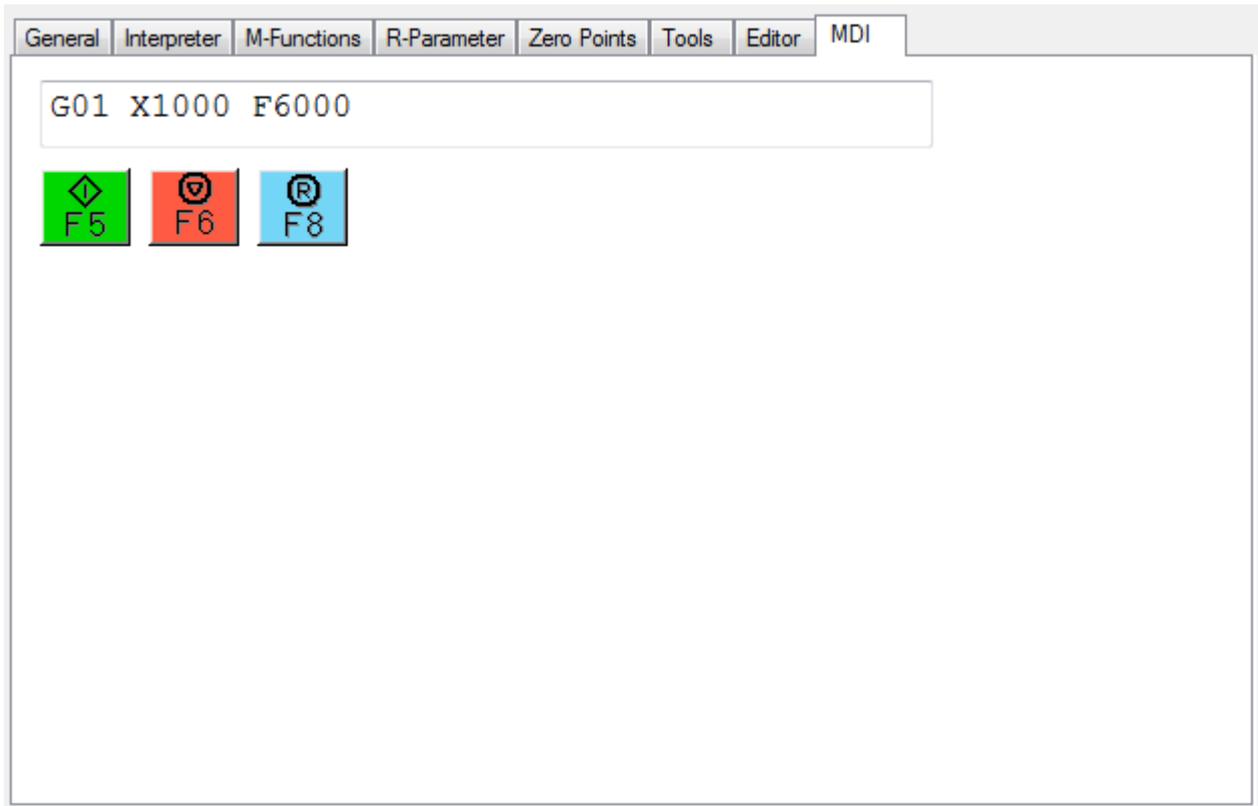
If the target system is connected via a remote connection, the NC-file has to be selected from the target system and cannot be loaded from the local machine.

- **F5**
Starts the currently loaded NC program.

i The NC program displayed in the editor does not necessarily have to be the currently loaded program.

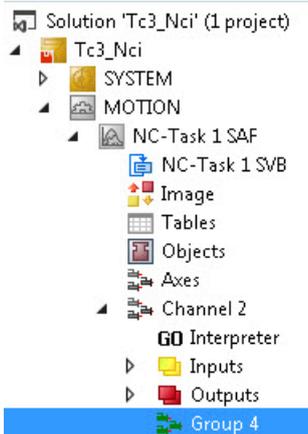
- **F6**
Stops the currently running NC program.
- **F7**
Loads the NC program displayed in the editor.
- **F8**
Executes a group reset.
- **F9**
Saves the NC program currently displayed in the editor under the same name.
- **Editor...**
Opens a larger window in which the NC program is displayed.

3.3.8 "MDI" tab



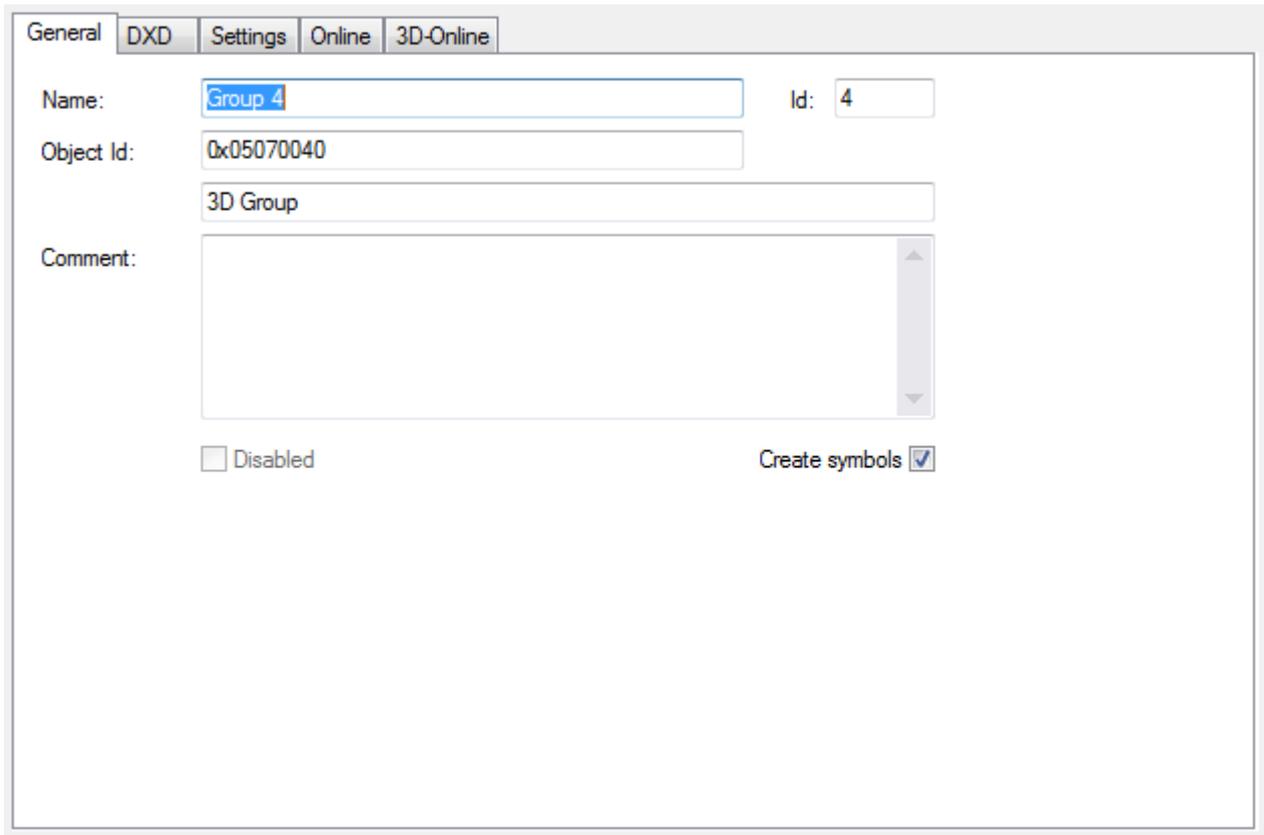
MDI stands for “Manual Data Interface”. It can be used to enter individual NC blocks directly from the TwinCAT 3 Engineering environment (XAE). Processing is started and stopped via F5 and F6 respectively.

3.4 Group element



General [▶ 24]
DXD [▶ 24]
Settings [▶ 28]
Online [▶ 29]
3D-Online [▶ 30]

3.4.1 "General" tab



The screenshot shows the 'General' tab of a configuration window. It contains the following fields and controls:

- Name:** Group 4
- Id:** 4
- Object Id:** 0x05070040
- 3D Group:** 3D Group
- Comment:** (Empty text area)
- Disabled
- Create symbols

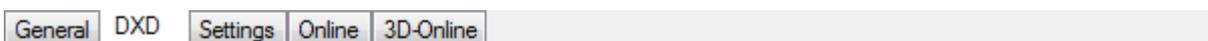
group ID

The group ID is shown on the "General" page. This is required for group-specific ADS commands.

Create symbols

In order to be able to access path variables symbolically, select symbol generation for the group here.

3.4.2 "DXD" tab



The screenshot shows the 'DXD' tab of a configuration window. The 'Settings' sub-tab is selected.

Parameter	Offline Value	Online Value	Type	Unit
Curve Velocity Reduction Mode	'COULOMB' ▼		E	
Velocity Reduction Factor for C0-Transition	0.1		F	
Velocity Reduction Factor for C1-Transition	1.0		F	
Critical Angle for Segment Transition 'Low'	10.0		F	°
Critical Angle for Segment Transition 'High'	75.0		F	°
Minimum velocity at segment transitions	0.0		F	
Global Soft Position Limits (for x,y,z-axes)	TRUE ▼		B	
Interpreter Override Type	Reduced ▼		E	
Enable calculation of the total remaining chord length	FALSE ▼		B	
Maximum number of transferred jobs per nc cycle [1 ... 20]	1		D	
SAF cycle time divisor	1		D	
User defined SAF table length [128 ... 1024]	128		D	

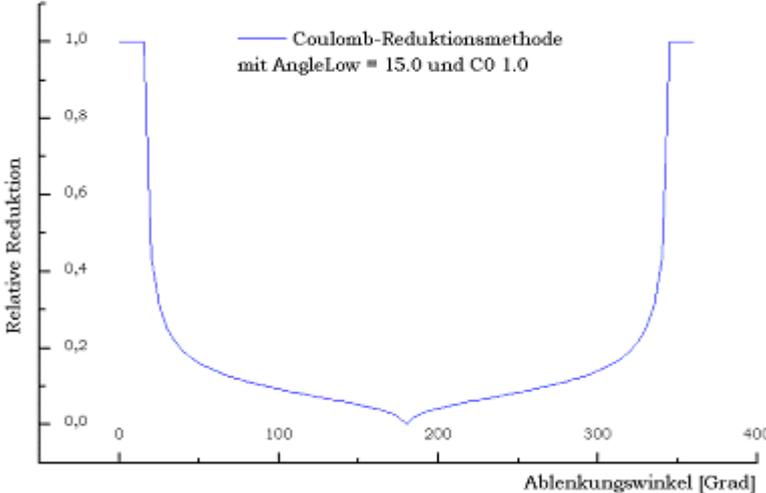
The NCI group parameters are written on the "DXD" properties page.

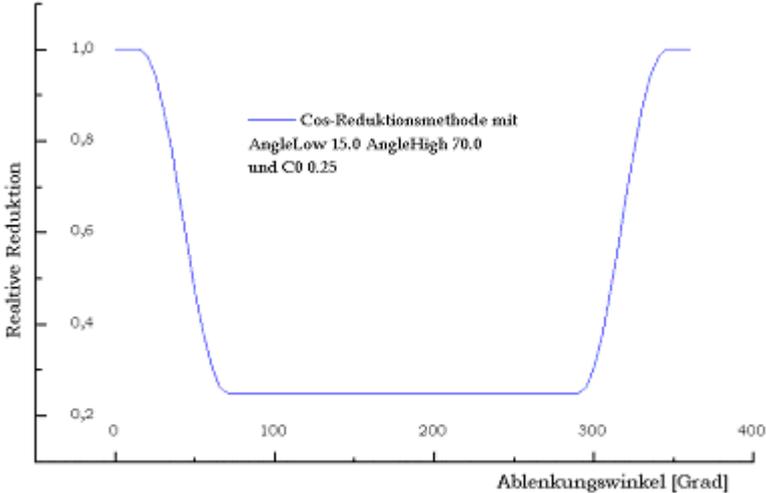
Curve velocity reduction method

The curve velocity reduction method is only effective for C0 transition (see [Classification of Segment Transitions \[▶ 319\]](#))

Defines of the curve velocity reduction method

- 0 Coulomb
- 1 Cosinus
- 2 VeloJump
- 3 DeviationAngle (not yet released)

Method	Description
Coulomb	<p>The coulomb reduction method is a dynamic process analogous to the Coulomb scattering. The deflection angle φ in the transition point is the angle between the tangents of the path at the end of the segment S1 and the tangent of the path at the start of segment S2. The velocity is set to the velocity at infinity, in analogy to Coulomb scattering, $V_k \propto (\tan(0.5(\pi-\varphi)))^{1/2}$ and then reduced via the C0 factor. $V_k \leftarrow C0 V_k$. In the case of a motion reversal ($\varphi=180$) the reduction is always $V_k = C0$. As the reduction in the case of small deflection angles is drastic, there is an angle $\varphi_{low} \in [0, 180]$ from which full reduction takes effect. To avoid reduction, set $\varphi_{low} = 180$. For full reduction (down to $\varphi = 0$), set $C0 = 0.0$ and $\varphi_{low} = 0$.</p> 

Method	Description
Cosine	<p>The cosine reduction method is a purely geometrical process. It involves:</p> <ul style="list-style-type: none"> the C0 factor $\in [0, 1]$, an angle $\varphi_{low} \in [0, 180]$, an angle $\varphi_{high} \in [0, 180]$ with $\varphi_{low} < \varphi_{high}$ <p>Reduction scheme:</p> <ul style="list-style-type: none"> $\varphi < \varphi_{low}$: no reduction: $V_k \leftarrow V_k$, $\varphi_{high} < \varphi$: reduction by the C0 factor: $V_k \leftarrow C0 V_k$ $\varphi_{low} < \varphi < \varphi_{high}$: partial reduction continuously interpolating between cases 1 and 2, proportional to the cos function in the range $[0, \pi/2]$. <p>For full reduction (down to $\varphi = 0$), set $C0 = 0.0$ and $\varphi_{low} = 0$ and φ_{high} very small but not equal to 0 (e.g. $1.0E-10$)</p> 
VeloJump	<p>It is a geometrical procedure for determining the segment transition velocity at a C0 transition. The procedure reduces the path velocity as required, so that the step change in velocity does not exceed the specified limit value. It is calculated based on the following formula: $VeloJump \text{ factor} * \text{cycle time} * \min(\text{acceleration; deceleration})$. Further information: ▶ 319</p>

Velocity reduction factor C0 transition

Reduction factor for C0 transitions. The effect depends upon the reduction method.

$C0 \in [0.0, 1]$

Velocity reduction factor C1 transition

First, V_{link} is set to the lower of the two segment target velocities:

$V_{link} = \min(V_{in}, V_{out})$.

The geometrically induced absolute step change in acceleration $AccJump$ in the segment transition is calculated depending on the geometry types G_{in} and G_{out} , and the plane selection G_{in} and G_{out} of the segments to be connected, at velocity V_{link} .

If this is greater than $C1$ times the path acceleration/(absolute) deceleration $AccPathReduced$ permissible for the geometries and planes, the velocity V_{link} is reduced until the resulting step change in acceleration is equal to $AccPathReduced$.

If this value is less than V_{min} , then V_{min} takes priority.

Note When changing the dynamic parameters, the permissible path acceleration for the geometries and planes and thereby the reaction of the reduction changes automatically.

Reduction factor for C1 transitions: $C1 \geq 0.0$

Critical angle, segment transition 'low'

Parameters for φ_{low} (see [curve velocity reduction method](#) [► 25]).

Critical angle, segment transition 'high'

Parameters for φ_{high} (see [curve velocity reduction method](#) [► 25]).

Minimum velocity at segment transitions

Each NCI group has a minimum path velocity $V_{min} \geq 0.0$. The actual velocity should always exceed this value. User-specified exceptions are: programmed stop at segment transition, path end and override requests which lead to a velocity below the minimum value. A systemic exception is a motion reversal.

With the reduction method DEVIATIONANGLE the deflection angle is $\varphi \geq \varphi_h$, in which case the minimum velocity is ignored. V_{min} must be less than the set value for the path velocity (F word) of each segment.

The minimum velocity can be set to a new value $V_{min} \geq 0.0$ in the NC program at any time. The unit is *mm/sec*.

Global soft position limits (for x,y,z-axes)

Parameters for enabling the software end positions of the path (see: [Parameterization](#) [► 321]).

Interpreter override type

Parameter for selecting the path override type (see [Path override \(interpreter override types\)](#) [► 322]).

Enable calculation of the total remaining chord length

Activates the calculation of the remaining path length. When the calculation of the remaining path length has been activated, it can be extracted via ADS afterwards. See also within the Appendix: [Displaying the Remaining Path Length](#) [► 318].

Maximum number of transferred jobs per nc cycle [1 ... 20]

Maximum number of commands to be transferred per NC cycle. With this parameter it is possible that the SVB task still runs slower than the SAF task and nevertheless sufficiently enough jobs are transposed so that the SAF table does not run out of jobs.

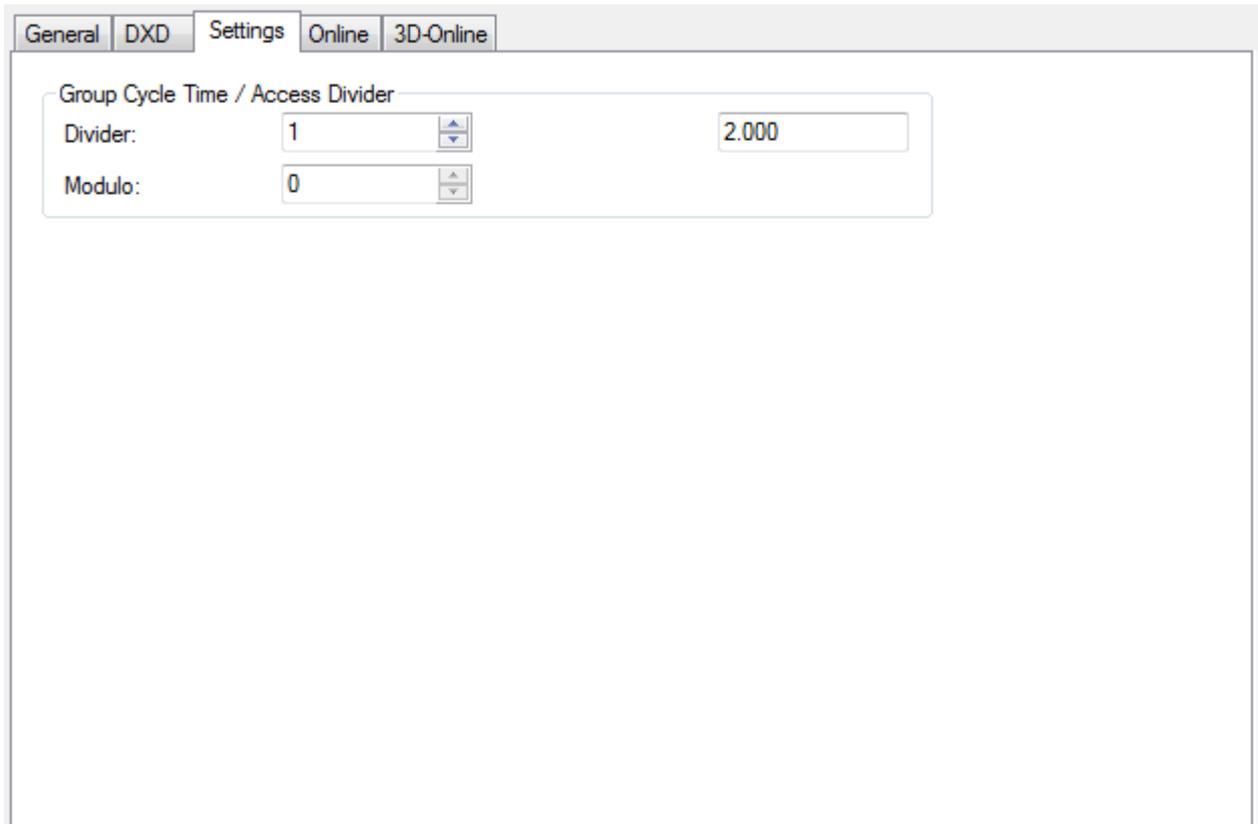
SAF cycle time divisor

The cycle time reduction ensures that the set value in the SAF is not calculated with the SAF cycle time, but with a time that is divided by the value specified here. For highly dynamic motions it may make sense to set the parameter to a value greater than 1, in order to minimize discretization inaccuracies. Increasing the SAF cycle time divisor results in the set value generator being called more frequently internally.

User-defined SAF table length

Parameter that defines the size of the SAF table and therefore the maximum number of cached SAF entries (look-ahead). If an NC program involves sequential movement of many very short segments, increasing this value can help to avoid an unintentional velocity reduction at the segment transitions.

3.4.3 “Settings” tab



Under the "Settings" tab you can set the cycle time for the interpolation. The cycle time set here is a multiple of the cycle time of the SAF task.

● Using the cycle time in the "Settings" tab

i The cycle time setting can be used if you have to select a cycle time for the interpolation that differs from the SAF task. Generally, the cycle time of the SAF task should be adjusted to set the cycle time.

3.4.4 "Online" tab

Field	Value
Error Code:	0 (0x0)
SVB-State:	Ready
SAF-State:	Idle
SVB Entries:	0
SAF Entries:	0

Error code

The current error code for the channel is displayed here. The value is the same as the value displayed in the online window of the interpreter under '[channel status \[► 14\]](#)'

SVB status

SVB status displays the current block preparation status (SVB = **Satzvorbereitung**). Possible SVB states are:

```
ERROR
IDLE
READY
START
DRIVEOUT
CALIBRATE
MFUNC
SYNCREC
DELAY
MFUNCWAIT
SPINDLEWAIT
```

PLC evaluation of the SVB status is normally not necessary.

SAF status

SAF status displays the current block execution status (SAF = **Satzausführung**). Possible SAF states are:

```
ERROR
IDLE
CONTROL
RUN
RUN_DRIVEOUT
WAIT
```

PLC evaluation of the SAF status is normally not necessary.

SVB entries

Number of current SVB entries.

SAF entries

Number of current SAF entries.

3.4.5 "3D-Online" tab

	Nominal Assignment	Actual Assignment	
X:	X	X	Clear
Y:	Y	Y	Clear
Z:	Z	Z	Clear
Q1:	(none)	(none)	Clear
Q2:	(none)	(none)	Clear
Q3:	(none)	(none)	Clear
Q4:	(none)	(none)	Clear
Q5:	(none)	(none)	Clear

Accept Assignment

Clear Assignment

Target assignment

At this point the **interpolation group** is formed. The movement of the PTP axes, which are assigned to the path axes X, Y and Z, can then be based on interpolation.

Any PTP axes can be selected with the aid of the selection lists for the path axes X, Y and Z. Press the 'Apply' button to form the 3D group.

A comparably PLC function block is available in the [PLC Library: Tc2_NCI \[▶ 195\]](#). (See [CfgBuildExt3DGroup \[▶ 196\]](#))

Actual assignment

The current path axis configurations are displayed here. Use 'Delete' to remove individual axes from the 3D group.

Delete whole configuration

Resolves the complete 3D group. Here, too, a corresponding PLC function block is available in the [PLC Library: Tc2_NCI \[▶ 195\]](#). (See [CfgReconfigGroup \[▶ 198\]](#))

4 GST Reference Manual

4.1 General Notes

All GST-examples in this documentation presuppose the following assumptions:

- Initially, the tool is located at `X0, Y0, Z0`.
- All state-variables of the interpreter are set to their default values, except that the velocity is set to a nonzero value.

4.2 Preprocessor

Include Directive

```
#include "<path>"  
#include < <path> >
```

The `#include` directive inserts the contents of another file. The included file is referenced by its path. Typically, it is used to “import” commonly used code like e.g. libraries. Its behavior is similar to the C-Preprocessor.

Example:

In the following example file `a.nc` includes file `b.nc`. On execution of `a.nc` the interpreter internally replaces the include-line by the text of `b.nc`. Therefore, executing the program `a.nc` has the same effect as executing the program `c.nc`.

FILE a.nc:

```
G01 X0 Y0 F6000  
#include "b.nc"  
G01 X0 Y100
```

FILE b.nc:

```
G01 Z-2  
G01 X100  
G01 Z2
```

FILE c.nc:

```
G01 X0 Y0 F6000  
G01 Z-2  
G01 X100  
G01 Z2  
G01 X0 Y100
```

- If `path` is absolute, it is directly used to locate the included file. An absolute path must be surrounded by quotation marks.
- If `path` is relative and surrounded by quotation marks, it is appended to the directory of the including file to form the path of the included file.
- If `path` is enclosed in angle brackets, it is regarded to be relative to the paths in the `searchpath` list. The first entry in this list that leads to an existing file is used for inclusion. The `searchpath` list is supplied by the interpreter environment of the interpreter.

Example:

The following example assumes that the `searchpath` is set to the directories `c:\jjj` and `c:\kkk`. The file `aaa.nc` consists of a sequence of `#include`-directives that are explained in the following.

- The file `bbb.nc` is included using an absolute path. Therefore, its location is independent of the location of `aaa.nc`. Absolute referencing is useful for files that always reside at a fixed location on the filesystem.
- The file `ccc.nc` is referenced relative. It must reside in the directory of `aaa.nc` (the including file), which is `c:\mmm\`.
- The file `ddd.nc` is also referenced relative. It is expected to reside at `c:\mmm\ooo\ddd.nc`.
- The relative reference of `eee.nc` uses the sequence `'..'`, which refers to the parent directory. Therefore, the file `eee.nc` is expected in `c:\ppp\qqq\eee.nc`.
- The relative path of `fff.nc` is denoted in angle brackets. Therefore, the directories in the `searchpath` are considered, rather than the directory of `aaa.nc`. The file is expected in `c:\jjj\fff.nc` or `c:\kkk\fff.nc`. The first path that leads to an existing file is considered. If there is no file `fff.nc` in any directory of the `searchpath`, an error is reported.
- Finally, the file `ggg.nc` is expected in `c:\rrr\ggg.nc`. Both entries in the `searchpath` lead to this location.

FILE `c:\mmm\aaa.nc`:

```
#include "c:\nnn\bbb.nc"
#include "ccc.nc"
#include "ooo\ddd.nc"
#include "..\ppp\qqq\eee.nc"
#include <fff.nc>
#include <../rrr/ggg.nc>
```

- Each include-directive must be denoted on a dedicated line. Then, this entire line is replaced by the contents of the included file. An additional 'newline' character is appended.
- The include-directive may be used multiple times at arbitrary locations of the including file.
- If an included file does not exist, an error is reported.
- If the include directive is not placed at the first position of a line, an error is reported.
- Include directives in included files are also subject to replacement.
- An infinite loop due to recursive inclusion (e.g. A includes B, B includes C and C includes A) is detected and reported as an error.
- The same file may be included multiple times.



It is typically bad practice to include a file multiple times. Especially, if this feature is misused to factor out code. Instead, a function should be preferred to define code that is reused multiple times (see section [Userdefined Functions \[► 57\]](#)).

Example:

In the following example file `a.nc` includes file `b.nc` twice. The second inclusion is always expanded, independently of the enclosing condition by the `IF-THEN` expression. The included file `b.nc` itself includes file `c.nc`.

FILE `a.nc`:

```
G01 X100 F6000
#include "b.nc"
G01 Y100
! IF stVariable=47 THEN
#include "b.nc"
! END_IF;
```

FILE `b.nc`:

```
#include "c.nc"
G01 X0 Y0
```

FILE `c.nc`:

```
G01 Z0
```

Example:

File `x.nc` demonstrates a series of invalid include directives. The first three lines violate the rule that each include directive must be denoted on a dedicated line. In lines 4 and 5 the filename is not properly enclosed in quotation marks or angle brackets. In line 6 a nonexisting file is included. Line 7 violates the rule that the include directive always has to be placed at the first position of a line. Line 8 includes the file `y.nc`, which itself includes file `x.nc`. This loop is reported as an error.

FILE x.nc:

```
#include "a.nc" G01 X100
! #include "a.nc"
#include "a.nc"    #include "b.nc"
#include a.nc
#include "a.nc>
#include "non_existing_file.nc"
    #include "a.nc"
#include y.nc
```

FILE y.nc:

```
#include "x.nc"
```

4.3 Combining G-Code and ST

A GST-Program

```
<g-code>
<g-code>
! <st-code>
<g-code>
<g-code>
{
<st-code>
<st-code>
! <g-code>
<st-code>
<st-code>
}
<g-code>
<g-code>
```

A GST-file consists of sequences of G-code and sequences of ST-code that can be interleaved as shown above. Each program starts in G-code mode. The mode can be switched to ST for one line using an exclamation mark (!). The ST-mode ends at the end of line automatically.

As an alternative a block of ST-code can be defined using curly braces ('{...}'). This notation is more practical to define a long sequence of ST-code in a GST-program. Within the ST-block the G-code mode can be entered for one line using the exclamation mark. Thereby, the G-code mode ends at the end of line automatically.

G-Code Block

```
<address><value> <address>=<G-Expression> <address>{<ST-Expression>}
```

A line of G-code is called a **block**. It consists of a sequence of **words**. A word is a combination of an **address** (e.g. G or X) and a **value**. A value can be defined by a literal (e.g. 2.54), by a G-expression (e.g. 2*foo+1) or by an ST-expression (e.g. sin(foo**2)-1).

G-Code Expression

```
<address>=a+b-c*d/e
```

The result of the expression is used as the value of the word. The four basic arithmetic operations ('+', '-', '*', '/') can be used in a G-expression. They are evaluated as expected, i.e. all operations are left-associative and '*', '/' have a higher precedence than '+', '-'. Variables that have been declared in ST can also be used in a G-expression (with respect to their scope).

All computations are performed using type `LReal` (64-bit floating point according to IEEE 754). The value of an ST-variable is implicitly converted to type `LReal` according to the conversion rules of ST. If a type (e.g. `STRING`) cannot be converted, an error is reported.

● **RESTRICTION:**

i ST-variables that contain a number in their name (e.g. `x0`) cannot be used in a G-expression to avoid confusion with a G-Code like `X0`. This limitation does not apply to ST-expressions.

● **RESTRICTION:**

i Array variables, struct variables and objects cannot be used in a G-expression. This limitation does not apply to ST-expressions.

● **RESTRICTION:**

i Parentheses are not allowed in a G-expression as they are used to denote comments in G-Code. For the same reason function calls are not available. These limitations do not apply to ST-expressions.

Embedded ST-Expression

`<address>{<ST-Expression>}`

The result of the ST-expression is used as the value of the word. It must be convertible to `LReal`. Basically, an ST-expression is ST-Code that could be placed on the right hand side of an assignment. Other ST-Code (e.g. an ST-statement) is not allowed. However, extensive computations can be encapsulated in an ST-function that is then called in the ST-expression.

● **i** An ST-expression should not have side effects, since the evaluation order of ST-expressions is generally undefined and may change in the future. Besides, this style of programming employing side effects is a bad programming style. For instance, an ST-expression should not call a function that contains G-Code.

Example:

- The following GST-program starts with a line of G-code that moves the tool rapidly to the origin.
- The line is followed by a line of ST-code that declares variable 'i'. The ST-mode is entered by the prefixed exclamation mark ('!'). After this line G-code mode resumes.
- The G-code in line 3 moves the tool down.
- Lines 4 to 8 define a block of ST-code that contains a FOR-loop. The code in this block is interpreted as ST-code, except for the G-code line in line 6. This line of G-code uses a G-expression to set the X-axis to $10 \cdot i$. The value of the Y-axis is defined using an ST-expression that is enclosed in curly braces. This expression evaluates to 0 if 'i' is even and to 10 otherwise.
- The programmed path of the program is shown in Figure "ExampleExpressions".

```
G00 X0 Y0 Z0
! VAR i : INT; END_VAR
G01 Z-1 F6000
{
FOR i := 1 TO 5 DO
!G01 X=i*10 Y{ (i MOD 2) *10 }
END_FOR;
}
```

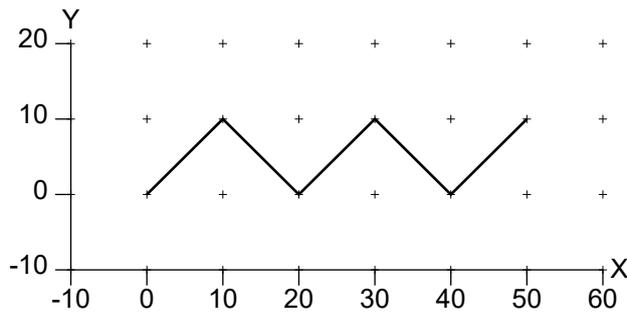


Figure "ExampleExpressions".

Suppression of G-Code Blocks

`/<n> <G-Code block>`

The execution of a G-Code block can be suppressed conditionally. If `'/<n>'` is prefixed and the n -th bit in an internal disable mask is set, the block is suppressed (not executed). The disable mask can be set by the PLC and by the ST-function `disableMaskSet`. If n is omitted, it has 0 value by default. [See section [Suppression of G-Code Blocks \[► 81\].](#)]

4.4 G-Code (DIN 66025)

4.4.1 Comments

DIN 66025 Comment

`<g-code> (<comment>) <g-code>`

Text that is enclosed in round parentheses is treated as comment in G-Code (according to DIN 66025). The comment must not include further parentheses. A comment within round parentheses can extend for multiple blocks or lines and therefore may skip a carriage return, too.

Example:

The following example demonstrates the notation of comments in G-Code.

```
N10 G01 X0 Y-10 F3000
N20 G01 (activate linear interpolation) X10 (set X-coordinate to
10) Y0 F6000
(the next block results in a semicircle with center point
X10 Y10)
N30 G02 (activate clockwise interpolation) Y20 U10 (radius is 10)
M02
```

Line Comment

`<g-code> // <comment>`

Text between `'//'` and the end of line is treated as a comment in G-Code.

Example:

The following example demonstrates the notation of line comments in G-Code.

```
N10 G01 X10 F6000 // perform a linear movement to X10 Y0
// the next block results in a semicircle with center point X10 Y10
N20 G02 Y20 U10
M02
```

4.4.2 Execution Order

A block (line of G-code) consists of a sequence of words. The programmed order of words is not considered by the GST interpreter. Instead, the following execution order is obeyed that consists of 7 sequential and dependent steps.

1. Reference System	N*	Set block number.
	G17..G19	Selection of a workingplane.
	G70, G71, G700, G710	Selection of a unit.
	G90, G91	Selection of absolute/ incremental programming.
	D*, P*	Selection of a tool and its orientation.
2. Configuration	G40..G42	(De-)activation of Tool Radius Compensation.
	G53..G59	Selection and programming of zero offset shift.
	F*	Set velocity.
3. M-Function Pre	M*	M-functions that are configured as "before".
4. Parameter to PLC	H*, S*, T*	
5. Movement	Q*, G00..G03	Movement to a point.
	G09, G60	Activation of accurate stop.
6. Wait	G04	Wait for a given duration.
7. M-Function Post	M*	M-functions that are configured as "after".

The first step sets up the reference system. The second step configures following movements. Note that the second step may depend on the first one. E.g. the programmed velocity (F) considers a velocity unit (G700) that is programmed in the same block. Step three and the following steps perform actions like a movement.

4.4.3 Mutual Exclusive G-Codes

Certain combinations of G-Codes must not be programmed in the same block (line of G-Code). Such conflicting G-Codes typically set state variables to contradictory values (e.g. set length unit to *mm* and to *inch*). There are also combinations that use the same parameters and therefore must not be programmed in the same block (e.g. G58 and G59). Below is a list of groups of G-Codes. G-Codes that belong to the same group are in conflict.

- G00, G01, G02, G03, G04, G58, G59
Interpolations and programmed zero-offset-shift.
- G70, G71, G700, G710
Set unit for length and speed.
- G90, G91
Set absolute/ relative programming.
- G53, G54, G55, G56, G57
Deactivate/ select zero-offset-shift.
- G40, G41, G42
Deactivate/ activate Tool Radius Compensation.
- G17, G18, G19
Select workingplane.

4.4.4 Rapid Traverse (G00)

Set the interpolation mode to “rapid, linear”. The interpolation mode applies to this block and all succeeding blocks until it is reset by G01, G02 or G03. G00 is the default interpolation mode.

If G00 is active, programming of a point (see X) will result in a linear geometry segment that is processed with maximum velocity. The programmed velocity is not considered. G00 is typically used to position the tool. For machining G01 should be used, which considers the programmed velocity.



G01, G02, G03, G04, G58 and G59 are mutually exclusive. They must not be programmed in a common block.

Example:

The resulting path of the following example is shown in Figure “ExampleG00”. The first block N10 rapidly moves the tool to position X20, Y10, Z30. The resulting geometry segment is a line in space. The orientation remains unchanged. The second block N20 performs a rapid movement to X50, Y10, Z30. There is no need to denote G00 in this line, since interpolation is modal.

```
N10 G00 X20 Y10 Z30
N20 X50
M02
```

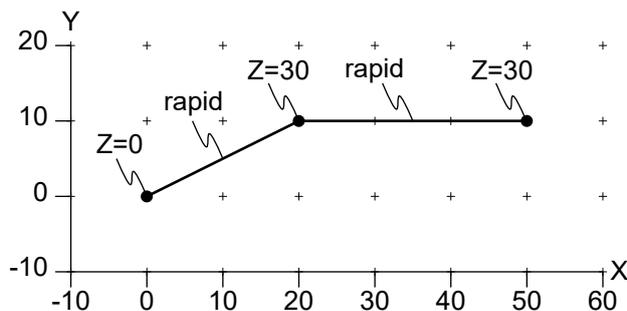


Figure “ExampleG00”.

4.4.5 Linear Interpolation (G01)

Set the interpolation mode to “linear”. This interpolation mode is like G00, except that the path is machined with the programmed velocity. (See F.) The interpolation mode applies to this block and all succeeding blocks until it is reset by G00, G02 or G03.

```
N20 G01 X100.1 Y200 F6000
N30 X150
M02
```

4.4.6 Circular Interpolation (G02, G03, IJK, U)

G02

Set the interpolation mode to “circular/helical, clockwise”. The interpolation mode applies to this block and all succeeding blocks until it is reset by G00, G01 or G03. If G02 is active, programming of a point will result in a circular (or helical) arc that is machined with the current velocity. (See General Codes (F, N, Q, X, Y, Z, A, B, C) [► 48].) In the following, a circular arc is regarded. The helical arc is covered later.

A circular arc starts at the current point and ends at the programmed point. It rotates around the working-plane normal (PCS, i.e. program coordinate system) in the center point. The center point can be defined using Centerpoint Programming or using Radius Programming.

Centerpoint Programming

For Centerpoint Programming the center is defined relative to the starting-point using the I, J, K parameters. The center point is the sum of the starting-point and the vector [I, J, K]. The I, J, K parameters are optional and have 0 value by default. If the starting-point and the endpoint are equal with respect to the workingplane, a full circle will be emitted.

● CONSTRAINT:
i The radius at the starting-point and at the endpoint must be equal. However, small deviations are allowed and corrected automatically. (See Centerpoint Correction [► 75].)

● CONSTRAINT:
i The center point must not be equal to the starting-point or endpoint.

Radius Programming

For Radius Programming the center point is derived from the radius that is given by the U parameter. Typically, there are two arcs of a given radius that lead from the starting-point to the endpoint. If the radius is positive, the shorter one is used, otherwise the longer one is chosen. Apart from that, the absolute value of the radius is regarded by the interpreter.

● CONSTRAINT:
i Radius Programming can by its nature not be used to program a full circle. This curvature can be programmed by Centerpoint Programming.

● CONSTRAINT:
i The radius must not be zero.

● CONSTRAINT:
i The radius must not be smaller than half of the distance between starting-point and endpoint with respect to the workingplane.

Helical

If the starting-point and endpoint do not lie in a plane that is parallel to the workingplane, a *helical movement* is performed.

TIP: moveCircle3D
i The ST-function `moveCircle3D` is a more powerful way to define a circle or helix. It covers 3D-arcs and multiturn circles.

Example:

The following example results in the path that is shown in Figure “ExampleG00G02”. The block N10 uses Radius Programming to define a clockwise arc from X0 Y0 to X10 Y10 with radius 10. Because the radius is positive, the center point c1 of the shorter arc is chosen. In block N30 the center point c2 of the longer arc is used because the radius is negative. The block N50 uses Centerpoint Programming, where the center c3=[60, 0, 0] is the sum of the starting-point [50, 0, 0] and [I, J, K]=[10, 0, 0]. The block N70 defines a full circle with center point C04 because the starting-point and endpoint are equal. The block N90 defines a helical arc with center point C05 and height 30 (in Z-direction).

```
N01 G00 X0 Y0
N10 G02 X10 Y10 U10 F6000
N20 G00 X30 Y0
N30 G02 X40 Y10 U-10
N40 G00 X50 Y0
N50 G02 X60 Y10 I10
N60 G00 X80 Y0
N70 G02 J10
N80 G00 X110 Y0
N90 G02 J10 X120 Y10 Z30
M30
```

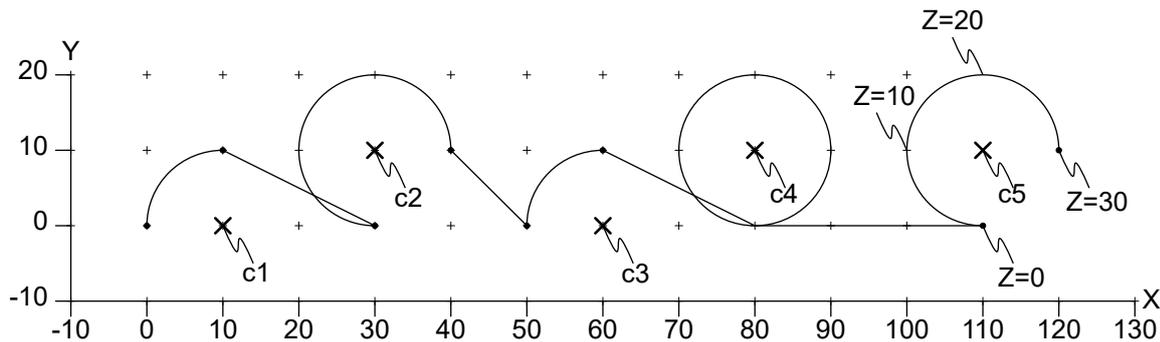


Figure “ExampleG00G02”.

G03

Set the interpolation mode to “circular/ helical, counterclockwise”. This interpolation behaves similar to G02. The interpolation mode applies to this block and all succeeding blocks until it is reset by G00, G01 or G02.

IJK

I<vx> J<vy> K<vz>

Defines the center point for circular movements. See G2, G3 for details. The center point is defined as `currentPoint + [vx, vy, vz]`. The current length unit is used for vx, vy, vz. The parameters I, J, K are optional and have a 0 default value.

U

U<v>

In the context of G2 or G3 the radius is set to $|v|$. The current length unit is used for v. If v is positive, the shorter arc is used to interpolate between the current and the next point. If v is negative, the longer one is used. See G2, G3 for details.

4.4.7 Dwell Time (G04)

G04

Suspend machining for a given duration. The duration is defined by either `X` or `F` in the current time unit. (See `unit` for details.)

Example:

The following example assumes that the current time unit is set to seconds. On one execution of the program the machine moves to `X10`, waits for 1.5 seconds and then moves to `X20`.

```
N10 G01 X10 F6000
N20 G04 F1.5
N30 G01 X20
M02
```

4.4.8 Accurate Stop (G09,G60)

G09

Accurate Stop - Nonmodal

G09 evokes an Accurate Stop.
G09 is nonmodal.

G60

Accurate Stop - Modal

G60 evokes an Accurate Stop.
G60 is modal. G00 calls off G60.

4.4.9 Delete Distance to go (G31)

G31 ("delete distance to go") is activated block by block via the NC program. This command enables deleting of the residual distance of the current geometry from the PLC with the function block `ltpDelDtgEx` [► 205]. In other words, if the command is issued while the block is processed, the motion is stopped with the usual deceleration ramps. The NC program then processes the next block. An error message is generated if the PLC command is not issued during the execution of a block with "delete distance to go" selected.

G31 always effects an implicit decoding stop, i.e. an exact positioning always occurs at the end of the block.

Example:

```
N10 G01 X0 Y0 F6000
N20 G31 G01 X2000
N30 G01 X0
N40 M02
```

Requirements

Development Environment	Target System
TwinCAT V3.1.4024.20	PC or CX (x86 or x64)

4.4.10 Zero Offset Shifts (G53,G54...59)

G53

Deactivate any zero offset shift translation. This adjustment is the default. The deactivation becomes active also for the current block. See sections [Zero Offset Shift \[► 82\]](#) and G58/ G59 for details.

G54..G57

Activates the translation that is associated with the given G-Code (TZ54...TZ57). Also activates the translations of G58 and G59. The translations apply to the current block and all succeeding blocks until changed. See section [Zero Offset Shift \[► 82\]](#) for details.

G58, G59

Set the translation that is associated with the given G-Code. The new translation value is given by the parameters X, Y, Z, which are mandatory. By default, the associated translations are zero. See section [Zero Offset Shift \[► 82\]](#) for details.

Example:

The resulting MCS (machine coordinate system) path and the applied translations of this example are shown in Figure "ExampleG54G58G59".

- The first line sets the translation that is associated with G54 to [0, 5, 0].
- The next line sets the programmed translation of G58 to [0, 10, 0]. Since zero-offset-shifts are still disabled (default G53), the PCS (program coordinate system) and MCS (machine coordinate system) match.
- Accordingly, the block N20 results in a linear movement from MCS (machine coordinate system) coordinate [0, 0, 0] to [20, 0, 0].
- The next line activates G54 and programs a linear movement along N30, whereby G54 becomes active before the movement. The programmed PCS (program coordinate system) coordinate [40, 0, 0] is mapped to the MCS (machine coordinate system) coordinate [40, 15, 0].
- The next line sets the programmed transformation G59 to [0, 5, 0]. Thereby, the effective translation changes from [0, 15, 0] to [0, 20, 0]. Since the current MCS (machine coordinate system) coordinate must not be affected by this change, the current PCS (program coordinate system) coordinate is set to [40, -5, 0], implicitly.
- The succeeding ST-function frameGet stores these coordinates in [pcsX, pcsY, pcsZ].
- The next line merely programs the x-coordinate of the end of segment N50. Therefore, the PCS (program coordinate system) coordinate of the end of segment N50 is [60, -5, 0], which is mapped to the MCS (machine coordinate system) coordinate [60, 15, 0]. In other words: The translation G59 is active, but does not become apparent due to the adaption of the current PCS (program coordinate system) coordinate. (See section [Applying Transformations \[► 96\]](#) for details.)
- It becomes apparent by the last line, which sets the PCS (program coordinate system) coordinate of the end of segment N60 to [80, 0, 0]. This coordinate is mapped to the MCS (machine coordinate system) coordinate [80, 20, 0].

```
!zeroOffsetShiftSet(g:=54, x:=0, y:=5, z:=0);
N10 G58 X0 Y10 Z0
N20 G01 X20 Y0 F6000
N30 G54 X40 Y0
N40 G59 X0 Y5 Z0
!VAR pcsX, pcsY, pcsZ : LREAL; END VAR
!frameGet(x=>pcsX, y=>pcsY, z=>pcsZ);
N50 X60
N60 X80 Y0
M02
```

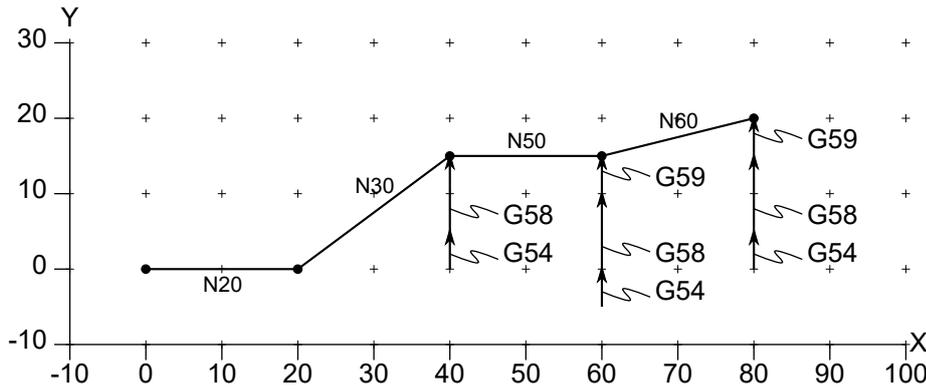


Figure "ExampleG54G58G59".

4.4.11 Tool Radius Compensation (D, G40, G41, G42)

D

D<v>

Select tool v. The new tool applies to its own block and all succeeding blocks until a new tool is selected. Tool 0 is special. Its selection deactivates any tool compensation. Tool 0 can be regarded as tool where all tool parameters are set to zero. It is selected by default.

Example:

In the following example tool 1 is defined to have a Y-offset of 10 and tool 2 to have an Y-offset of 20. Block N10 and block N50 use tool 0. Tool 1 applies to block N20 and to block N30. In block N40 tool 2 is active. Figure "ExampleD" shows the resulting programmed path (dotted line) and the resulting tool center point path (solid line).

```
!toolSet(index:=1, nr:=1, offsetY:=10);
!toolSet(index:=2, nr:=2, offsetY:=20);
N10 G01 X10 Y0 F6000
N20 G01 X20 Y0 D1
N30 G01 X30 Y0
N40 G01 X40 Y0 D2
N50 G01 X50 Y0 D0
M02
```

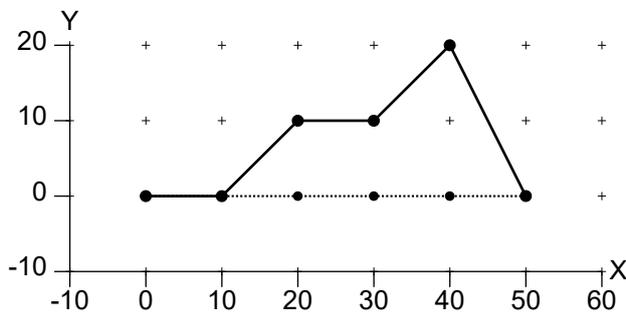


Figure "ExampleD".

G40

Deactivate Tool Radius Compensation (TRC).

G41

Activate tool radius compensation (TRC). After activation the programmed path is shifted left by the radius of the currently selected tool. (See D.)



On activation, a tool with a nonzero index must be selected.

Example:

The following example demonstrates the activation and deactivation of tool radius compensation. The programmed path (dotted line) and the compensated path (solid/ dashed line) are shown in Figure “ExampleG40G41”.

- The first line of the GST program sets the `offset` parameter to 5 mm. Therefore, the adjacent segments of a gap are extended by 5 mm. The remaining gap is closed by a circular arc.
- The second line defines the approach and depart behavior to use a circular arc with a radius of 5 mm and an angle of 90 degree.
- The third line defines tool 1 to have a radius of 10.
- Block N10 describes a linear movement to [10, 0, 0].
- The next block N20 selects tool 1 and activates tool radius compensation, where D1 comes into effect before G40 is processed and G40 is active before X20 is processed. Therefore, the end of segment N20 is subject to TRC (tool radius compensation). The linear movement from the end of segment N10 to the end of segment N20 in the programmed path is substituted by an approach-segment (dotted line) from the end of N10 to the end of N20' in the compensated path.
- In the next three lines a linear movement along N30, N40 and N50 is programmed. Since segment N40 would result in a collision, it is eliminated from the compensated path.
- In the next line a circular arc along N60 is programmed. The gap between the end of N50' and the beginning of N60' is closed as described earlier.
- The line along N70 is the last segment that is subject to TRC (tool radius compensation), since its deactivation becomes active before the end of N80. The line along N80 is replaced by the depart-segment N80', similarly to the approach-segment.

```
!trcOffsetSet(offset:=5);
!trcApproachDepartSet(approachRadius:=5, approachAngle:=90, departRadius:=5, departAngle:=90);
!toolSet(index:=1, tooltype:=tooltypeMill, radius:=10);
N10 G01 X10 F6000
N20 X20 G41 D1
N30 X35
N40 X40
N50 Y20
N60 G02 X50 Y10 U10
N70 G01 X70
N80 X80 Y0 G40
N90 X90
M02
```

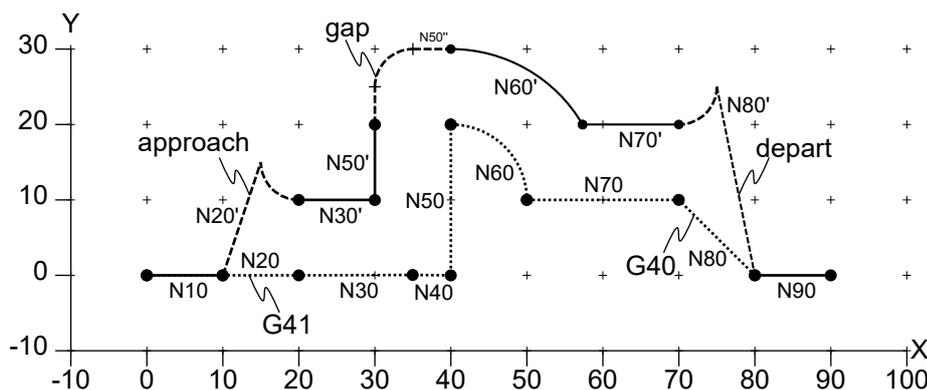


Figure “ExampleG40G41”.

G42

This function is the same as G41, except that the path is shifted to the right. See G41 for details.

4.4.12 Working Plane and Feed Direction (G17, G18, G19, P)

G17

Select XY-plane as workingplane, i.e. the workingplane normal is set to $[0, 0, 1]$. This workingplane is the default workingplane.

G18

Select ZX-plane as workingplane, i.e. the workingplane normal is set to $[0, 1, 0]$.

G19

Select YZ-plane as workingplane, i.e. the workingplane normal is set to $[1, 0, 0]$.

P

P<v>

Switch tool orientation. The value of v must be 1 or -1. If v is negative, the tool points in the direction of the working plane normal. Otherwise, it points into the opposite direction.

Example:

The resulting MCS-path (MCS: machine coordinate system) of the following example is shown in Figure “ExampleP”. The first line of the program defines Tool 1 to have a length of 10. G18 activates the XZ-workingplane.

```
N10:           The end of segment N10 is not subject to any tool compensation as D0 is active.

N20:           For segment N20 tool 1 is active with a positive tool orientation. To compensate the
                tool length the translation  $[0, 10, 0]$  is applied. (See section Transformations \[► 95\]
                for details.) Thereby, the PCS (program coordinate system) endpoint  $[20, 10, 0]$  of
                N20 is mapped to the MCS (machine coordinate system) endpoint  $[20, 20, 0]$ . The
                MCS (machine coordinate system) point and the applied transformation are shown in
                Figure “ExampleP”.

N30:           In block N30 the tool orientation is switched, which sets the translation to  $[0, -10, 0]$ .
                This translation is applied to the PCS (program coordinate system) endpoint of N30
                resulting in the MCS (machine coordinate system) endpoint  $[30, 0, 0]$ .

N20..N90:      The blocks N60..N90 are similar to N20..N50, except that the Y-coordinate is not
                programmed. Therefore, the tool length compensation does not become apparent,
                although it is active. That behavior happens because the current PCS (program
                coordinate system) point is always adapted on a changed transformation. (See section
                Applying Transformations \[► 96\] for details.)
```

```
!toolSet(index:=1, tooltype:=tooltypeDrill, length:=10);
G18
N10 X10 Y10 D0 F6000
N20 X20 Y10 D1
N30 X30 Y10 P-1
```

```
N40 X40 Y10 P1
N50 X50 Y10 D0
N60 X60 D1
N70 X70 P-1
N80 X80 P1
N90 X90 D0
M02
```

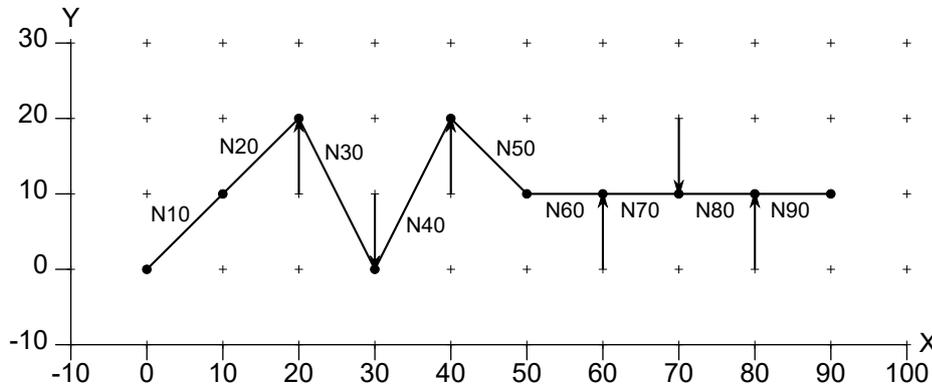


Figure "ExampleP".

4.4.13 Inch/metric dimensions (G70, G71, G700, G710)

G70

Set the unit for lengths to *inch*. The new unit also applies to the current block. G70 is equivalent to the call `unitLengthSet(unitLengthInch)`. The unit for velocity is not affected. See UnitLength and G71 for details.

G71

Set the unit for lengths to *millimeter*. The new unit also applies to the current block. G71 is equivalent to the call `unitLengthSet(unitLengthMillimeter)`. The unit for velocity is not affected. See UnitLength for details.

Example:

In Figure "ExampleG70G71" the path of the following example is shown, which uses the unit *millimeter*.

- The first line of the program sets the unit for lengths to *inch*. This unit is used in the same line to interpret X2 in inch. Thus, the path N10 ends at position [50.8 mm, 0 mm, 0 mm].
- Accordingly, the next line moves the tool along N20 towards [50.8 mm, 25.4 mm, 0 mm].
- The last line sets the unit to *millimeter*. Therefore, the path N30 ends at position [80 mm, 25.4 mm, 0 mm]. Accordingly, the segment N30 is a horizontal line.

```
N10 G01 X2 G70 F6000
N20 G01 Y1
N30 G01 X80 Y25.4 G71
M02
```

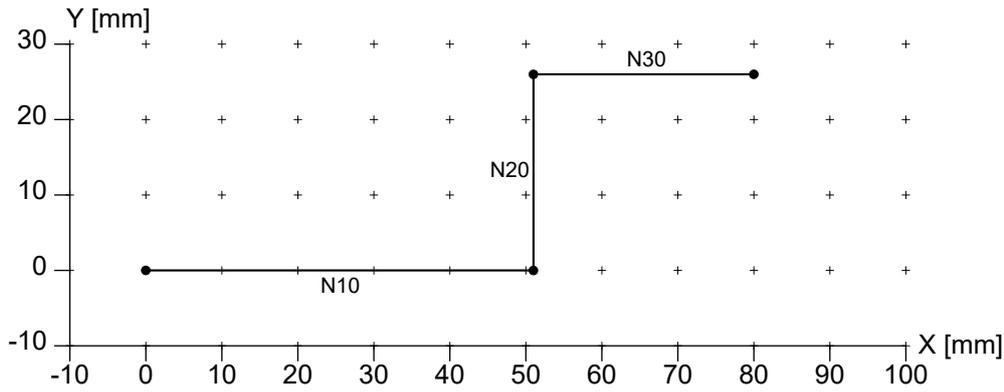


Figure "ExampleG70G71".

G700

Like G70, but also applies to the interpretation of velocity. The new unit comes into effect in the current block. G700 is equivalent to the calls `unitLengthSet (unitLengthInch)` and `unitVelocitySet (unitLengthInch,unitTimeMinute)`.

G710

Like G71, but also applies to the interpretation of velocity. The new unit comes into effect in the current block. G710 is equivalent to the calls `unitLengthSet (unitLengthMillimeter)` and `unitVelocitySet (unitLengthMillimeter,unitTimeMinute)`.

Example:

The path of the following example is shown in Figure "ExampleG700G710".

- The first line defines a linear movement to [1 in, 1 in, 0 in] with a velocity of 100 in/min.
- The second line sets the length unit to mm, but does not affect the velocity unit. It defines a movement to [30 mm, 10 mm, 0 mm] with a velocity of 50 in/min.
- The last line also sets the velocity unit to mm/min. Therefore, there is a movement to [40 mm, 20 mm, 0 mm] with a velocity of 1000 mm/min.

```
N10 G700 G01 X1 Y1 F100
N20 G71 G01 X50 Y10 F50
N30 G710 G01 X80 Y20 F1000
```

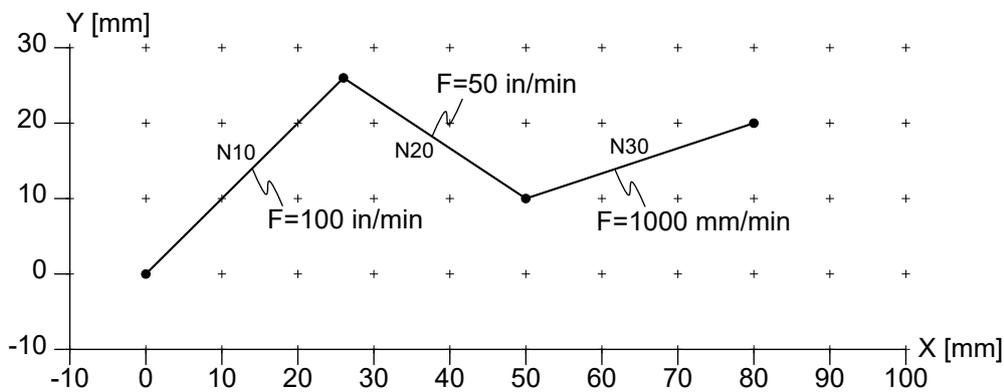


Figure "ExampleG700G710".

4.4.14 Dimensional Notation (G90, G91)

G90

Switches to absolute coordinates. X, Y, Z are interpreted as absolute PCS (program coordinate system) coordinates. This adjustment is the default. The switch becomes active in its own block.

G91

Switches to relative coordinates. X, Y, Z are interpreted to be relative to the current point, i.e. the next point is computed as the sum of [X, Y, Z] and the current point. The switch has an effect for its own block.

i **Implement Offsets Manually**

Using G91 and in this way switching to relative coordinates any Tool Offsets and Zero Shifts that have been defined earlier are not evaluated within these coordinates and therefore have to be defined and implemented manually within the framework of the G91-Code.

Example:

The path of the following example is shown in Figure “ExampleG90G91”. The switch to G90/ G91 takes effect immediately.

```
N10 G90 G01 X10 Y20 F6000
N20 X20 Y10
N30 G91 X10 Y10
N40 X10 Y-10
N50 G90 X50 Y20
M02
```

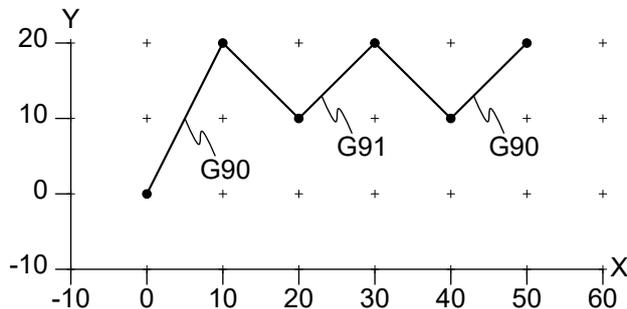


Figure “ExampleG90G91”.

4.4.15 M-Functions (M)

M

M<v>

Triggers the M-function v. The timing and behavior depends on the definition of v in the System Manager of TwinCAT.

M2 and M30 are internally defined. Both functions trigger a synchronization with the NC-channel. (See wait()-function, chapter [Synchronization](#) |▶ 78|.) Both functions stop the execution of the GST-program. Due to this order the interpreter waits for the completion of the NC-channel before it stops.

In addition, M30 also resets all fast M-functions and H, S, T.

i There must not be more than one M-function of type handshake in a block.

i The M-functions M2 and M30 do not have to be defined by the user in the System Manager of Twin-CAT.

Example:

This example assumes the following definitions of M-functions:

- M10: Fast before move.
- M11: Fast after move.
- M12: Fast before move, auto-reset, reset M10, M11.
- M20: Handshake before move.
- M21: Handshake after move.

Figure “ExampleM10M11M12M20M21” visualizes the programmed path and the activation of M-functions. The fast M-functions M10, M11 are reset by M12, which itself is reset automatically.

```
N10 G01 X10 F6000
N20 X30 M10 M20
N30 X50 M11 M21
N40 X70
N50 X90 M12
M02
```

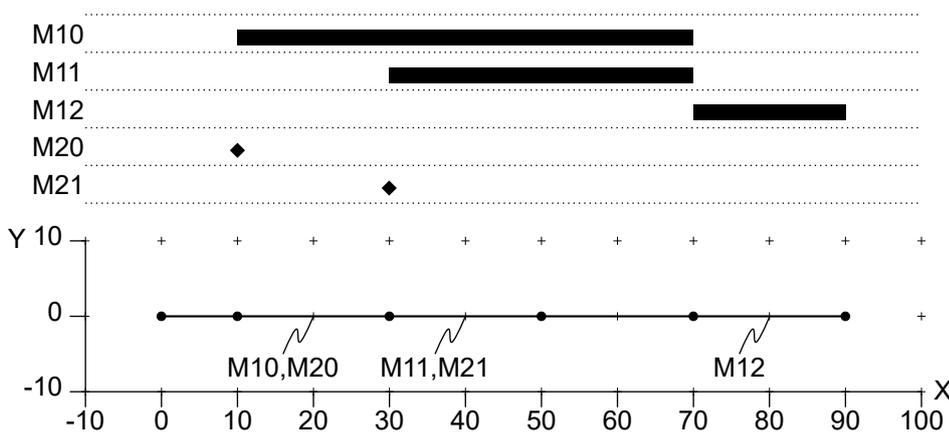


Figure “ExampleM10M11M12M20M21”.

4.4.16 General Codes (F, N, Q, X, Y, Z, A, B, C)

F

F<v>

Set velocity to v . Applies to the current block and all succeeding blocks until a new velocity is programmed. The unit for velocity selected currently is used. (See section [unitVelocitySet \[► 83\]](#) for details.) The default velocity is 0.



The velocity must be set to a nonzero value before a movement is programmed. Otherwise, an error is issued.

Example:

The first two segments N10 and N20 are processed with a velocity of 6000 mm/min, and the last segment N30 is processed with a velocity of 3000 mm/min.

```
N10 G01 X100 F6000
N20 G01 X200
N30 G01 X300 F3000
M02
```

N

N< v >

Set the block number to v . Typically, the block number is used to monitor the progress of the NC-program.

Q

Q< i >=< v >

Set the value of axis Q< i > to v where i must lie in the range 1 to 5. The Q-axes use linear interpolation.



The address letters Q and R are handled in a special way for historical reasons.



The address Q< i > has to be followed by a G-expression or by an ST-expression. The G-word Q1100 is invalid. Use Q1=100, instead.

Example:

The path of the following example is shown in Figure “ExampleQ”. The Q-axes are interpolated linear with the interpolation of a movement. The last block (N40) results in a linear interpolation of a Q-axis without a concurrent movement.

```
N10 G01 X30 Y0 Q1=100 F6000
N20 G02 X50 Y20 I20 Q2=200
N30 G01 X60 Q1=300 Q2=300
N40 Q1=0
M02
```

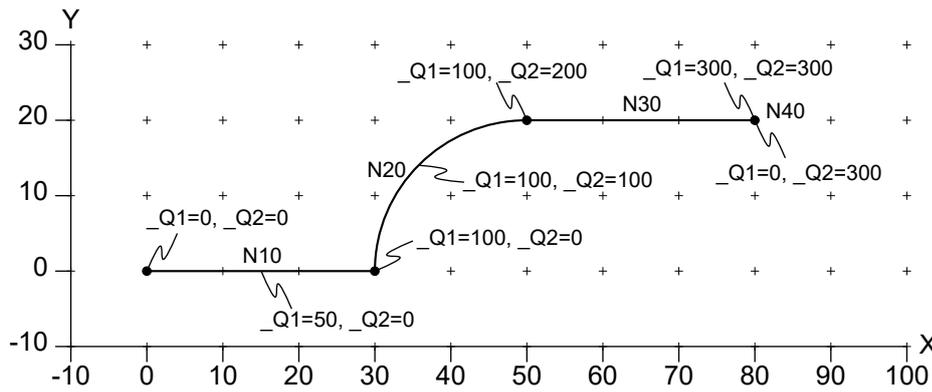


Figure "ExampleQ".

X

X<v>

Sets the X-coordinate of the next point to v. The current length unit is used for v.

Y

Y<v>

Sets the Y-coordinate of the next point to v. The current length unit is used for v.

Z

Z<v>

Sets the Z-coordinate of the next point to v. The current length unit is used for v.

A

A<v>

Sets the A-angle of the next orientation to v. The current angle unit is used for v.

B

B<v>

Sets the B-angle of the next orientation to v. For v the current angle unit is used.

C

C<v>

Sets the C-angle of the next orientation to v. For v the current angle unit is used.

4.5 ST - Structured Text (IEC 61131-3)

4.5.1 Comments

Line Comment

```
<st-code> // <comment>
```

Text between ‘//’ and the end of line is treated as comment in ST-Code.

Example:

```
{
VAR
  i : INT; // this variable is primarily used in FOR-loops for counting
END_VAR
}
```

/* */ Comment

```
<st-code> /* <comment>
<comment> */ <st-code>
```

Text between ‘/*’ and ‘*/’ is treated as comment in ST. This type of comment may be nested up to a depth of 3. The ‘/*...*/’-style comment may appear anywhere between literals, keywords, identifiers and special symbols. It may also contain G-Code lines.

Example:

The following example demonstrates the notation of comments in ST-Code. The first comment is placed within a variable declaration. The second comment encloses an entire ST-loop. The comment contains further comments and a G-Code line, which itself contains a G-Code comment.

```
{
VAR i /* used for counting */ : INT;  END_VAR

/* The following loop is commented out.
FOR i := 0 TO 10 DO
  /* zigzag pattern */
  ! G01 (linear interpolation) X=i Y{i MOD 2} F6000
  // end of loop
END_FOR;
*/
}
```

(* *) Comment

```
<st-code> (* <comment>
<comment> *) <st-code>
```

Text between ‘(*)’ and ‘*)’ is treated as comment in ST. This type of comment may be nested up to a depth of 3. It is similar to the ‘/*...*/’-style comment.

4.5.2 Literals

Integer Literals

Decimal	18
Binary	2#10010

Octal	8#22
Hexadecimal	16#12

The same integer value in decimal, binary, octal and hexadecimal notation.

Real Literals

Notation of real values

```
1.0
1.602E-19
```

Boolean Literals

Notation of Boolean values

```
0
1
TRUE
FALSE
```

Typed Literals

```
<typename>#<literal>
```

Typed literals where typename is a native type (e.g. `Word` or `LReal`) or an enumeration type (to avoid ambiguities).

Typing of literals is typically not necessary in GST, since the interpreter implements a decent typesystem that handles untyped literals properly. There are a few exceptions where the type of a literal is significant for semantics, like in the following example.

Example:

The first assignment assigns the value `16#80` to `w`, whereas the second one assigns the value `16#8000` to `w`.

```
{
VAR w: word; END_VAR
w := ror(BYTE#1,1);
w := ror(WORD#1,1);
}
```

String Literals

```
"abc"
'abc'
```

Notation of a 2-byte and a 1-byte string, respectively. Note that there is no implicit conversion between both types. The following escape-sequences can be used within both types of literals:

\$L	line feed
\$N	newline

\$P	form feed
\$R	carriage return
\$t	tab
\$' or \$"	quotes
\$<2 or 4 hexadecimal digits>	character of given code

Duration Literals

T#[+/-]<value><unit>[...]<value><unit>

TIME#[+/-] <value><unit>[...]<value><unit>

LT#[+/-]<value><unit>[...]<value><unit>

LTIME#[+/-]<value><unit>[...]<value><unit>

Time literals of type `TIME` or `LTIME`. The literal consists of an optional sign (+/-) and a sequence of `value/unit` pairs. `Value` must be an integer, except for the last one that may also be a floating point number. `Values` must not be negative and may be arbitrarily large. `Units` must appear in the following order.

d	day
h	hour
m	minute
s	second
ms	millisecond
us	microsecond
ns	nanosecond

An arbitrary subset of `units` may be used in a literal. For instance, the literal `T#1d15ms1500.01us` is valid.

Date Literals

DATE#<yyyy>-<mm>-<dd>

D#<yyyy>-<mm>-<dd>

LDATE#<yyyy>-<mm>-<dd>

LD#<yyyy>-<mm>-<dd>

Date literal of type `DATE` or `LDATE`. The literal is interpreted as UTC, i.e. timezone, daylight saving time and leap seconds are not considered. The year must not be smaller than 1970. The values `yyyy`, `mm` and `dd` have to be integer values, i.e. `D#1980-20-10` is a valid date literal, for example.

Time-of-Day Literals

`TIME_OF_DAY#<hh>:<mm>:<ss>`

`TOD#<hh>:<mm>:<ss>`

`LTIME_OF_DAY#<hh>:<mm>:<ss>`

`LTOD#<hh>:<mm>:<ss>`

Time-of-day literal of type `TOD` or `LTOD`. The literal is interpreted as UTC, i.e. timezone, daylight saving time and leap seconds are not considered. `hh` and `mm` must be integer values. `ss` may be an integer or a floatingpoint number, i.e. `TOD#7:30:3.1415` is a valid literal, for example.

Date-and-Time Literals

`DATE_AND_TIME#<yyyy>-<mm>-<dd>-<hh>:<mm>:<ss>`

`DT#<yyyy>-<mm>-<dd>-<hh>:<mm>:<ss>`

`LDATE_AND_TIME#<yyyy>-<mm>-<dd>-<hh>:<mm>:<ss>`

`LDT#<yyyy>-<mm>-<dd>-<hh>:<mm>:<ss>`

Date-and-time literal of type `DT` or `LDT`. The literal is interpreted as UTC, i.e. timezone, daylight saving time and leap seconds are not considered. This literal is a combination of the date literal and the time-of-day literal. Analogously, the corresponding rules for these two parts apply.

4.5.3 Native Data Types

Bitstring Types

`BOOL`, `BYTE`, `WORD`, `DWORD`, `LWORD`

Bitstring types of 1, 8, 16, 32 and 64 bit. Implicit conversion from left to right using zero extension.

Unsigned Integer Types

`USINT`, `UINT`, `UDINT`, `ULINT`

Unsigned integer types of 8, 16, 32 and 64 bit. Implicit conversion from left to right preserving the value.

Signed Integer Types

`SINT`, `INT`, `DINT`, `LINT`

Signed integer types of 8, 16, 32 and 64 bit. Implicit conversion from left to right preserving the value. An unsigned type of n bit is also implicitly converted to a signed type of m bit where the relation $m > n$ must hold. There is no implicit conversion between bitstring types and integer types.

Floating Point Types

`REAL`, `LREAL`

Floating point data types of 32 and 64 bit. Implicit conversion from left to right preserving the value.

String Types

```
string[<length>]
```

```
wstring[<length>]
```

1-byte and 2-byte strings of given length. If length is omitted, it has 255 as default value.

Character Types

```
char
```

```
wchar
```

Single 1-byte and 2-byte character of a string. It can be implicitly converted to a string.

Time-Related Types

```
TIME, LTIME
```

```
DATE, LDATE
```

```
TIME_OF_DAY, TOD, LTIME_OF_DAY, LTOD
```

```
DATE_AND_TIME, DT, LDATE_AND_TIME, LDT
```

Datatypes for duration, date and time. Internally, all values of these types are represented with a granularity of 1 nanosecond. Values of date-related types represent the number of nanoseconds since 1.1.1970 (UTC). Leapseconds are ignored. Implicit conversion is allowed from a non-L type to an L type, e.g. from TIME to LTIME.

4.5.4 Userdefined Types

Derived Types

```
TYPE
```

```
<typeName>: <typeName> := <defaultValue>;
```

```
END_TYPE
```

Definition of a new type as an alias to an existing type. The default value is optional.

Enumeration Types

```
TYPE
```

```
<typeName> : (<enumValue>, ..., <enumValue>) := <defaultValue>;
```

```
END_TYPE
```

Definition of an enumeration type. The default value is optional.

Enumeration Types with Defined Values

```
TYPE
```

```
<typeName> : (<enumValue>:=<integer value>, ...,  
<enumValue>:=<integer value>) := <defaultValue>;
```

```
END_TYPE
```

Definition of an enumeration type with user-defined values for each element. The default value is optional.

Array Types

```
TYPE
```

```
<typeName>: ARRAY [<from>..<>to>,<from>..<>to>] OF <typeName> :=
[<defaultValue>, <repetition>(<defaultValue>), ...];
```

```
END_TYPE
```

Definition of an array type. The array may be multi-dimensional. The index range is defined for each dimension. At runtime the boundaries of the array are checked. A boundary violation leads to a runtime-error. The default values are defined in ascending order starting with the last dimension. A value can be repeated by placing it into parentheses prefixed with the number of repetitions. If the number of defined default values does not match the array size, initialization is truncated or padded with the default value of the element type. In either cases a compile-time warning is issued.

Structure Types

```
TYPE
```

```
<typeName>: STRUCT
    <memberName>: memberType;
    ...
END_STRUCT := (<memberName> := <defaultValue>, ...);
```

```
END_TYPE
```

Defines a structure type of the given members. Currently, the default value is placed after the type definition. This positional style is a difference to the ST-standard.

Pointer Types

```
TYPE
```

```
<typeName>: REF_TO <basetypeName>;
```

```
END_TYPE
```

Defines a pointer type of the given base type.

4.5.5 Control Structures

IF-THEN-ELSIF-ELSE

```
IF <condition> THEN
    <statements>
ELSIF <condition> THEN
    <statements>
ELSE
    <statements>
END_IF;
```

Conditional statement. The ELSIF-branch and ELSE-branch are optional. ELSIF can be repeated arbitrarily.

CASE OF

```
CASE <expression> OF
    <value>, <value>, ..., <value>: <statements>
ELSE
    <statements>
END_CASE;
```

The case-list consists of a comma-separated sequence of values or ranges. Only the first matching case is executed. The optional ELSE-branch is executed if no case matches.

FOR

```
FOR <variable> := <expression> TO <expression> BY <expression> DO
    <statements>
END_FOR;
```

Iterates over the given `variable` in the defined range (including) using the supplied step-size. If the latter is omitted, it has 1 as default value.

WHILE

```
WHILE <condition> DO
    <statements>
END_WHILE;
```

Pre-checked loop.

REPEAT

```
REPEAT
    <statement>
UNTIL <condition>
END_REPEAT;
```

Post-checked loop. The break condition is evaluated after performing the `<statements>` the loop includes.

EXIT

```
EXIT;
```

EXIT can be used within loops to leave the loop. If loops are nested, only the innermost loop is left. If there is no loop surrounding the EXIT keyword, a compile-time error is issued.

4.5.6 Userdefined Functions

Function Definition

```
FUNCTION <name> : <returntype>
VAR_INPUT
    <variable declarations>
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
    <variable declarations>
```

```
END_VAR
```

```
VAR_IN_OUT
```

```
    <variable declarations>
```

```
END_VAR
```

```
VAR
```

```
    <variable declarations>
```

```
END_VAR
```

```
VAR_EXTERNAL
```

```
    <variable declarations>
```

```
END_VAR
```

```
    <statements>
```

```
END_FUNCTION
```

Declares a function. Thereafter, it is callable by its name. The declaration of the return type is optional. If it is supplied, the function returns a value of the given type. The return value is defined within the function body by an assignment to the function name.

The function may have input, output and in-out parameters. The order of declaration is significant. It is used for nonformal calls. Declared variables are only used within the function body. External variables are imported from global scope. Variables and parameters are not persistent, i.e. they do not retain their value between two calls.

Nonformal Function Call

```
<functionname>(<expression>, ..., <expression>)
```

Nonformal function call. The order of expressions must match the number and order of declared parameters.

Formal Function Call

```
<functionname>(
    <inputParamName> := <expression>,
    <outputParamName> => <variableName>,
    <inputParamName> := <variableName>)
```

Formal function call. Parameters are identified by their name. If a declared parameter is not listed, it is implicitly set to its default value.



Do not Mix Formal with Nonformal

Mixing formal with nonformal function calls leads to invalid GST-syntax.

4.5.7 Standard Functions

4.5.7.1 Type Conversion

Type Conversion (*_TO_*)

`<nativeType>_to_<nativeType>(x)`

`to_<nativeType>(x)`

Explicit conversion between the given native types. The second alternative is overloaded for any applicable type.

For conversion from floatingpoint to integer x is rounded.

4.5.7.2 Arithmetic and Trigonometric

ABS

`ABS(x)`

Returns the absolute value of x .

The function is overloaded for any integer type and floatingpoint type. The type of x is used as return type.

SQRT

`SQRT(x)`

Returns the square root of x .

The function is overloaded for any floatingpoint type. The type of x is used as return type.



RESTRICTION:

Variable x must not be negative.

LN

`LN(x)`

Returns the natural logarithm of x , i.e. the logarithm to the base e .

The function is overloaded for any floatingpoint type. The type of x is used as return type.



RESTRICTION:

Variable x must be larger than 0.

LOG

`LOG(x)`

Returns the logarithm of x to the base 10.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

**RESTRICTION:**

Variable x must be larger than 0.

EXP

EXP (x)

Returns e raised to the power of x .

The function is overloaded for any floatingpoint type. The type of x is used as return type.

SIN

SIN (x)

Returns the sine of x where x is expected to be in radians.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

See also: The gSin function (chapter [Trigonometric \[► 84\]](#)).

COS

COS (x)

Returns the cosine of x where x is expected to be in radians.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

See also: The gCos function (chapter [Trigonometric \[► 84\]](#)).

TAN

TAN (x)

Returns the tangent of x where x is expected to be in radians.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

See also: The gTan function (chapter [Trigonometric \[► 84\]](#)).

ASIN

ASIN (x)

Returns the arc sine of x within the interval $[-\pi/2, \pi/2]$ radians.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

See also: The gASin function (chapter [Trigonometric \[► 84\]](#)).

**RESTRICTION:**

Variable x must lie within the interval $[-1, 1]$.

ACOS

ACOS (x)

Returns the arc cosine of x within the interval $[0, \pi]$ radians.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

See also: The gACos function (chapter [Trigonometric](#) [► 84]).

**RESTRICTION:**

Variable x must lie within the interval $[-1, 1]$.

ATAN

ATAN (x)

Returns the arc tangent of x within the interval $[-\pi/2, \pi/2]$ radians.

The function is overloaded for any floatingpoint type. The type of x is used as return type.

See also: The gATan function (chapter [Trigonometric](#) [► 84]).

ATAN2

ATAN2 (y, x)

Returns the arc tangent of y/x within the interval $[-\pi, \pi]$ radians.

The function is overloaded for any floatingpoint type. The smallest common type of x and y is used as return type.

See also: The gATan2 function (chapter [Trigonometric](#) [► 84]).

ADD

ADD (x_1, x_2, \dots)

Returns the sum of all parameters. The ADD-function can have an arbitrary number of parameters, but has to have at least one.

The function is overloaded for any integer and floatingpoint type. The smallest common type of all parameters is used as return type.

MUL

MUL (x_1, x_2, \dots)

Returns the product of all parameters. The MUL-function can have an arbitrary number of parameters, but has to have at least one. The infix-operator '*' can be used as an alternative.

The function is overloaded for any integer and floatingpoint type. The smallest common type of all parameters is used as return type.

SUB

SUB (x, y)

Returns the difference $x-y$. The infix-operator '-' can be used as an alternative.

The function is overloaded for any integer and floatingpoint type. The smallest common type of x and y is used as return type.

DIV

DIV (x, y)

Returns the quotient x/y . The infix-operator '/' can be used as an alternative.

The function is overloaded for any integer and floatingpoint type. The smallest common type of x and y is used as return type. If the return type is an integer type, the result is truncated towards zero.

**RESTRICTION:**

Variable y must not be zero.

MOD

MOD (x, y)

Returns the remainder of the integer division x/y . The infix-operator 'MOD' can be used as an alternative.

The function is overloaded for any integer type. The smallest common type of x and y is used as return type. The result may also be negative. The equation $x = \text{MUL}(\text{DIV}(x, y), y) + \text{MOD}(x, y)$ holds.

**RESTRICTION:**

Variable y must not be zero.

EXPT

EXPT (x, y)

Returns x raised to the power of y .

The function is overloaded such that x has a floatingpoint type and y has a floatingpoint type or integer type. The type of x is used as return type, i.e. returned is a Real or an LReal floating point type. The infix-operator '**' can be used as an alternative.

**RESTRICTION:**

If x is negative, then y must be an integer.

**RESTRICTION:**

If x is zero, then y must be larger than zero.

4.5.7.3 Shift and Rotation**SHL**

SHL (x, y)

Returns the bitstring x shifted left by y bits. Zero-bits are inserted at the right side. The least significant bit is assumed to be rightmost.

The function is overloaded for any bitstring type for x and any integer type for y . The type of x is used as return type.

**CONSTRAINT:**

Variable y must not be negative.

SHR

SHR (x, y)

Returns the bitstring x shifted right by y bits. Zero-bits are inserted at the left side. The least significant bit is assumed to be rightmost.

The function is overloaded for any bitstring type for x and for any integer type for y . The type of x is used as return type.

**CONSTRAINT:**

Variable y must not be negative.

ROL

$\text{ROL}(x, y)$

Returns the bitstring x rotated left by y bits. Bits that are shifted out at the left side are inserted at the right side. The least significant bit is assumed to be rightmost.

The function is overloaded for any bitstring type for x and for any integer type for y . The type of x is used as return type.

**CONSTRAINT:**

Variable y must not be negative.

ROR

$\text{ROR}(x, y)$

Returns the bitstring x rotated right by y bits. Bits that are shifted out at the right side are inserted at the left side. The least significant bit is assumed to be rightmost.

The function is overloaded for any bitstring type for x and for any integer type for y . The type of x is used as return type.

**CONSTRAINT:**

Variable y must not be negative.

4.5.7.4 Logical Operations

AND

$\text{AND}(x1, x2, \dots)$

Returns the bitwise Logical And of all parameters. Bit i is set in the result if bit i is set in all parameters. The **AND** function can have an arbitrary number of parameters, but has to have at least one.

The function is overloaded for any bitstring type. The smallest common bitstring type is used as return type.

OR

$\text{OR}(x1, x2, \dots)$

Returns the bitwise Logical Or of all parameters. Bit i is set in the result if bit i is set in at least one of all parameters. The **OR** function can have an arbitrary number of parameters, but has to have at least one.

The function is overloaded for any bitstring type. The smallest common bitstring type is used as return type.

XOR

$\text{XOR}(x1, x2, \dots)$

Returns the bitwise Logical Exclusive Or of all parameters. Bit i is set in the result if bit i is set in an uneven number of all parameters. The `XOR` function can have an arbitrary number of parameters, but has to have at least one.

The function is overloaded for any bitstring type. The smallest common bitstring type is used as return type.

NOT

`NOT(x)`

Returns the bitwise complement of x . Bit i is set in the result if bit i is not set in x .

The function is overloaded for any bitstring type. The type of x is used as return type.

4.5.7.5 Selection (Conditional Expressions)

SEL

`SEL(cond, x1, x2)`

Returns $x1$ if `cond` is false, and $x2$ otherwise.

MUX

`MUX(select, x0, x1, ..., xN)`

Returns $x<select>$. If `select` is 0, $x0$ is returned. If `select` is 1, $x1$ is returned and so forth. The `MUX` function can have an arbitrary number of parameters, but has to have at least two.

The function is overloaded for any type for $x<i>$ and for any integer for `select`. The smallest common type of $x<i>$ is used as return type.



RESTRICTION:

The variable `select` must lie within the interval $[0, N]$. Otherwise, an out-of-bounds error is issued at runtime.

4.5.7.6 Min, Max and Limit

MAX

`MAX(x1, x2, ...)`

Returns the maximum of all parameters.

The function is overloaded for any integer and floatingpoint type. The smallest common type of all parameters is used as return type.

MIN

`MIN(x1, x2, ...)`

Returns the minimum of all parameters.

The function is overloaded for any integer and floatingpoint type. The smallest common type of all parameters is used as return type.

LIMIT

`LIMIT(min, in, max)`

Returns `in` if it lies in the interval $[min, max]$. Otherwise, the violated bound (`min` or `max`) is returned.

The function is overloaded for any integer and floatingpoint type. The smallest common type of all parameters is used as return type.

**CONSTRAINT:**

The `min` boundary must be smaller than the `max` boundary.

4.5.7.7 Comparison

GT

GT (`x`, `y`)

Returns `TRUE` if `x` is larger than `y`. The smallest common type of `x` and `y` is used to perform the comparison.

The function is overloaded for all integer and floatingpoint types. The returntype is `BOOL`.

GE

GE (`x`, `y`)

Returns `TRUE` if `x` is not smaller than `y`. The smallest common type of `x` and `y` is used to perform the comparison.

The function is overloaded for all integer and floatingpoint types. The returntype is `BOOL`.

EQ

EQ (`x`, `y`)

Returns `TRUE` if `x` and `y` are equal. The smallest common type of `x` and `y` is used to perform the comparison.

The function is overloaded for all integer and floatingpoint types. The returntype is `BOOL`.

LE

LE (`x`, `y`)

Returns `TRUE` if `x` is not larger than `y`. The smallest common type of `x` and `y` is used to perform the comparison.

The function is overloaded for all integer and floatingpoint types. The returntype is `BOOL`.

LT

LT (`x`, `y`)

Returns `TRUE` if `x` is smaller than `y`. The smallest common type of `x` and `y` is used to perform the comparison.

The function is overloaded for all integer and floatingpoint types. The returntype is `BOOL`.

NE

NE(`x`,`y`)

Returns `TRUE` if `x` and `y` are not equal. The smallest common type of `x` and `y` is used to perform the comparison.

The function is overloaded for all integer and floatingpoint types. The returntype is `BOOL`.

4.5.8 R-Parameters

Arithmetic Parameters

The arithmetic parameters, for short known as R-parameters, are interpreter variables that are named by an expression of the form “R<n>”. Since ‘n’ is an integer in the range 0 . . 999, a total of 1000 R-parameters are available. The first 900 values R0 . . R899 of these are local variables for the NC channel. They can only be accessed by the interpreter of the channel. The R-parameters R900 . . R999 are declared globally. They exist only once for each NC, and all channels access the same storage. This kind of accessibility organization makes it possible to exchange data (e.g. for part tracing, collision avoidance etc.) beyond channel boundaries.

Assigning a Value to an R-Parameter

Assigning a value to an R-parameter is merely possible within Structured Text. There are two ways of assigning a value to an R-parameter. The value can be assigned directly or the rSet function can be employed. The function rSet is suitable to use when the index of the R-parameter to be assigned should not be determined until runtime.

Structured Text: Assigning an R-Parameter Value Directly

```
R<n> := LReal;
```

Example

```
!R1 := 7;
```

Structured Text: Assigning an R-Parameter Value with the “rSet” Function

```
rSet(index := LINT, value := LREAL)
```

Example

```
!rSet(1, 7);
```

Reading an R-Parameter Value

There are two ways of reading an R-parameter. An R-parameter can be used in G-Code directly or it can be extracted within Structured Text using the rGet function. The function rGet extracts an R-parameter value according to its index.

Structured Text: Reading an R-Parameter Value with the “rGet” Function

```
rGet(index := LINT) : LREAL
```

G-Code Example: Extracting an R-Parameter Value Directly

```
!R1 := 7;
N10 G01 X=R1 F6000
```

G-Code Example: Extracting an R-Parameter Value with the “rGet” Function

```
!R1 := 7;
N10 G01 X={rGet(1)} F6000
```

Example: Assigning and Extracting

```
{
VAR
    valueR1 : LREAL;
END_VAR

rSet(1, 7);
valueR1 := rGet(1);

R2 := 10;
R3 := R1 + R2;

!N10 G01 X=R1 Y0 Z=R2 F6000
!N20 G01 X={rGet(3)}
```

```
MSG(toString('R1 = ', valueR1, ',R2 = ', rGet(2), ', R3 = ', R3));
}
M02
```

Output:

```
R1 = 7.000000, R2 = 10.000000, R3 = 17.000000
```

● R-Parameters in Subroutines (Functions)

i Within a subroutine (function) an R-parameter has to be declared via a `VAR_EXTERNAL` declaration.

Example:

```
{
FUNCTION myFunction : LREAL
VAR_EXTERNAL
    R45: LREAL;
END_VAR
}
```

```
N10 G01 X=R45 F6000
```

```
!END_FUNCTION
```

Requirements

Development Environment	Target System
TwinCAT V3.1.4024.4 or 4022.32	PC or CX (x86 or x64)

4.6 CNC Functions

4.6.1 Strings and Messages

The toString-Function

```
toString(<arg0>, ..., <argN>): STRING
```

Converts and concatenates the given arguments to one string. This string is limited to 255 characters, which is the default string length. The `toString`-function behaves like the `print` function, except that it yields a formatted string instead of printing.

i The `toString`-function is especially useful to format a string for the `msg(...)`-function.

Msg

```
msg(str:= String[81])
```

Send the given message to the message list of TwinCAT. The message is processed by the NC-channel synchronously. It appears in the user-interface when all preceding NC-commands are completed.

To send formatted strings this function can be combined with the `toString`-function.

i The message is restricted to 81 characters. Text exceeding this restriction will be truncated.

Example:

The path of the following example is shown in Figure “ExampleMsg”. It is annotated with the emitted messages.

```
{
VAR
  x,y,z: LREAL;
  start: LDT;
END_VAR

!N10 G00 X0 Y0 F300
start := currentLdt();
!N20 G01 X30
msg('N20 completed');
!N30 X60 Y10
frameGet(x=>x,y=>y,z=>z);
msg(toString('Current position: [' ,x,',',y,',',z,']'));
!N40 X90
sync();
msg(toString('Machining time: ', currentLdt()-start));
}
M02
```

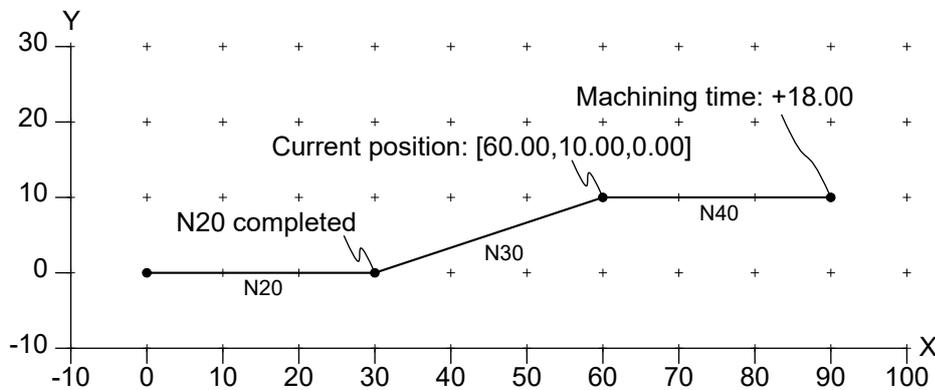


Figure “ExampleMsg”.

4.6.2 Transformations

transRotX/Y/Z

```
transRotX(angle:= LReal)
```

```
transRotY(angle:= LReal)
```

```
transRotZ(angle:= LReal)
```

Rotation around the respective axis by the given angle in the user-defined angle unit. The rotation is pushed onto the stack of transformations. The angle value is interpreted using the current angle-unit. See section [Transformations \[► 95\]](#) for details.

Example:

The resulting path of the following example is shown in Figure “ExampleTransRotZ”.

- N10 is programmed with the PCS (program coordinate system) and the MCS (machine coordinate system) being equal.
- N20 is programmed after a 45-degree rotation around the z-axis in $[0, 0, 0]$ has been pushed onto the stack of transformations. Another rotation of 45 degrees is pushed onto the transformation stack such that the rotations add up to 90 degree.
- Therefore, the MCS (machine coordinate system) coordinate of the end of segment N30 is $[0, 30, 0]$.

```
N10 G01 X30 Y0 F6000
!transRotZ(45);
N20 G01 X30 Y0
!transRotZ(45);
N30 G01 X30 Y0
!transPop();
!transPop();
M02
```

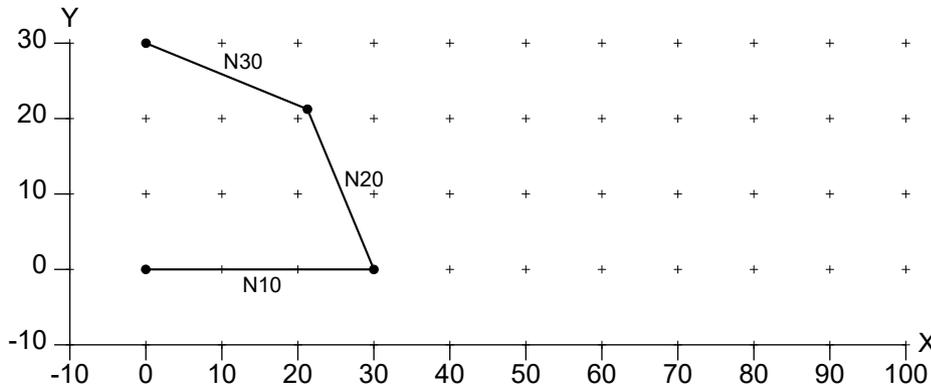


Figure “ExampleTransRotZ”.

transRotA

```
transRotA(x:=LReal, y:=LReal, z:=LReal, angle:=LReal)
```

Rotate around vector $[x, y, z]$ by the given angle. The rotation is pushed onto the stack of transformations. The angle value is interpreted using the current angle-unit. See section [Transformations \[► 95\]](#) for details.

i The vector $[x, y, z]$ must not be the zero vector.

Example:

The resulting path of the following example is shown in Figure “ExampleTransRotA”. The first invocation of `transRotA` rotates the PCS (program coordinate system) around the positive z-axis (right-hand rule) by 45 degree. The second invocation rotates around the negative z-axis by the same angle, i.e. into the opposite direction. The combination of both rotations is the identity transformation.

```
!transRotA(0,0,1,45);
N10 G01 X30 Y0 F6000
!transRotA(0,0,-1,45);
N20 G01 X30 Y0
!transPop();
!transPop();
M02
```

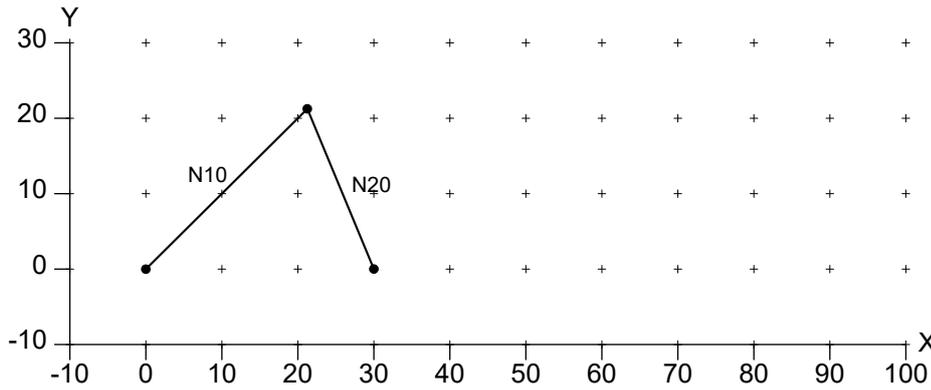


Figure "ExampleTransRotA".

transMirrorX/Y/Z

transMirrorX()

transMirrorY()

transMirrorZ()

Mirror with respect to the X-direction, Y-direction or Z-direction relative to the origin of the current PCS (program coordinate system). The transformation is pushed onto the stack of transformations.



The invocation of a mirror function switches the orientation of the coordinate system from right-handed to left-handed or vice versa. Most notably, this behavior switches the rotation direction of circles and the compensation direction of tool radius compensation. By default, the coordinate system is right-handed.

Example:

The resulting path of the following example is shown in Figure "ExampleTransMirrorX". The PCS (program coordinate system) is mirrored along the X-dimension. Thereby, the coordinate system becomes a left-handed system, within which the rotation direction of G2 is (intentionally) swapped.

```
N10 G02 X20 Y20 U20 F6000
!transMirrorX();
N20 G02 X-40 Y0 U20
!transPop();
M02
```

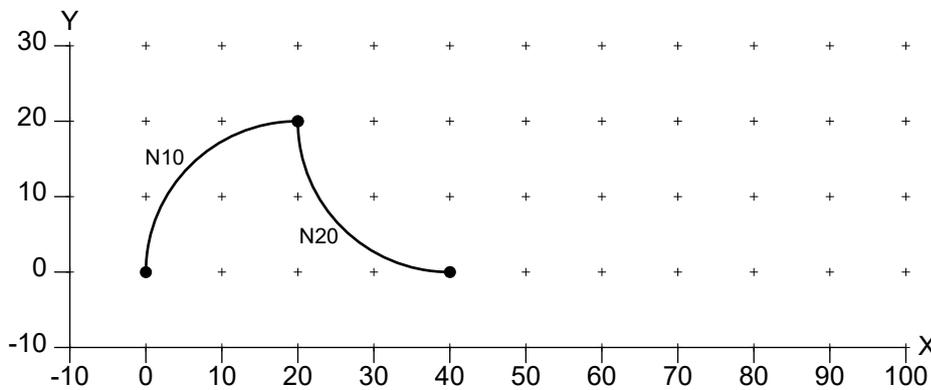


Figure "ExampleTransMirrorX".

transScale

```
transScale(factor:= LReal)
```

Scales the coordinate system by the `factor` in the X-dimension, Y-dimension and Z-dimension. The transformation is pushed onto the stack of transformations.

i The factor must be nonzero.

i If the factor is negative, the coordinate system is effectively mirrored in the X-dimension, Y-dimension and Z-dimension. Thus, the orientation of the coordinate system is swapped.

Example:

The resulting path of the following example is shown in Figure “ExampleTransScale”. After scaling by a factor of 2, the endpoint of segment N20 is mapped to [60, 20, 0].

```
N10 G01 X30 Y10 F6000
!transScale(2);
N20 G01 X30 Y10
!transPop();
M02
```

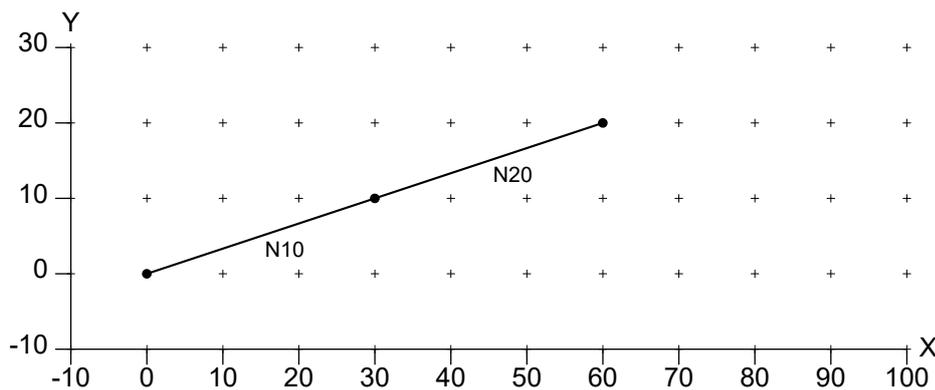


Figure “ExampleTransScale”.

transScaleAxis

```
transScaleAxis(axisNo := axisIndex, factor := value);
```

Scales the selected path axis (`axisNo`) by the factor. The supported axis and indexes are:

- X: 0
- Y: 1
- Z: 2

Q-axes are not supported.

i A different axes scaling is only allowed for linear movements, not for circular movements.

Example 1

```
N10 G01 X30 Y10 F6000
!transScaleAxis(axisNo:= 0, factor:=2.0);
!transScaleAxis(axisNo:= 1, factor:=2.0);
```

```
!transScaleAxis(axisNo:= 2, factor:=3.0);
N20 G01 X30 Y10
N30 G03 X40 Y10 I5 J0
M02
```

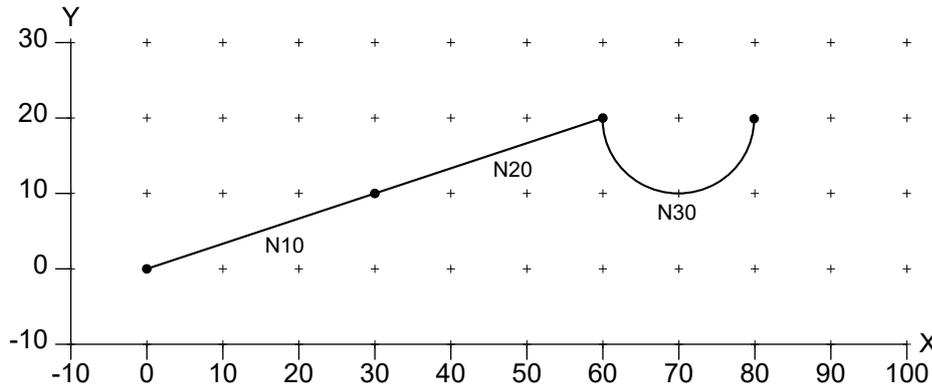


Figure "Example 1 TransScaleAxis".

Example 2

```
N10 G01 X20 Y5 F6000
!transScaleAxis(axisNo:= 0, factor:=2.0);
!transScaleAxis(axisNo:= 1, factor:=2.0);
!transScaleAxis(axisNo:= 2, factor:=3.0);
N20 G01 X20 Y5
!transScaleAxis(axisNo:= 0, factor:=2.0);
!transScaleAxis(axisNo:= 1, factor:=3.0);
N30 G01 X20 Y5
M02
```

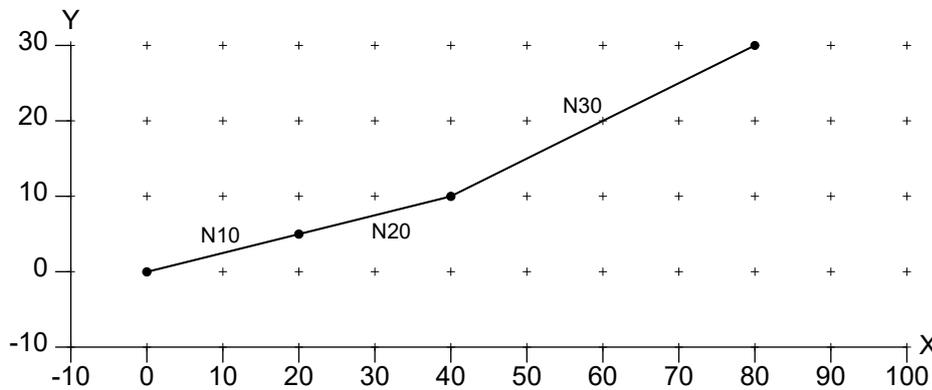


Figure "Example 2 TransScaleAxis".

Requirements

Development Environment	Target System
TwinCAT V3.1.4024.20	PC or CX (x86 or x64)

transTranslate

```
transTranslate(x:=LReal, y:=LReal, z:=LReal)
```

Translate by vector $[x, y, z]$. The translation is pushed onto the stack of transformations.

Example:

The resulting path of the following example is shown in Figure "ExampleTransTranslate". After translating by $[40, 20, 0]$ the endpoint of segment N20 is mapped to $[80, 20, 0]$.

```
N10 G01 X20 Y0 F6000
!transTranslate(40,20,0);
N20 G01 X40 Y0
!transPop();
M02
```

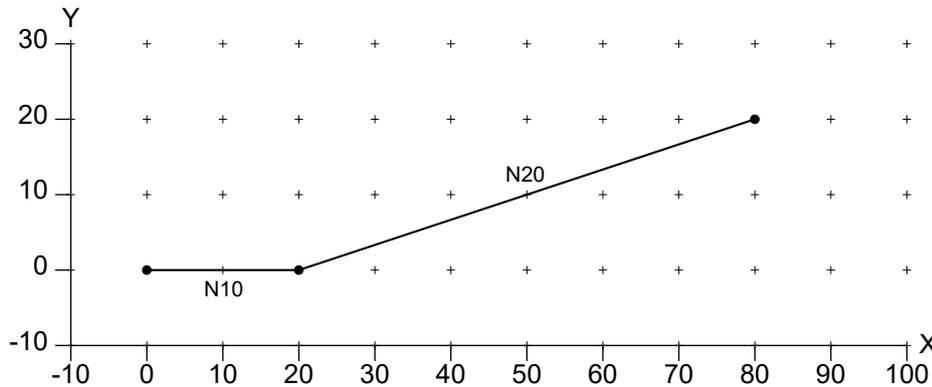


Figure "ExampleTransTranslate".

transPop

```
transPop()
```

Pops a transformation from the stack of transformations.

Example:

The resulting path of the following example is shown in Figure "ExampleTransPop". This example pushes the translation $[0, 20, 0]$ onto the stack, followed by the translation $[0, 10, 0]$. Thereby, the effective translation for N30 is $[0, 30, 0]$. The invocation of `transPop` removes the translation $[0, 10, 0]$ from the stack. Thus, the endpoint of segment N40 is translated by $[0, 20, 0]$. After removing the last translation from the stack the endpoint of segment N50 is not translated at all.

```
N10 G01 X10 Y0 F6000
!transTranslate(0,20,0);
N20 G01 X30 Y0
!transTranslate(0,10,0);
N30 G01 X50 Y0
!transPop();
N40 G01 X70 Y0
!transPop();
N50 G01 X90 Y0
M02
```

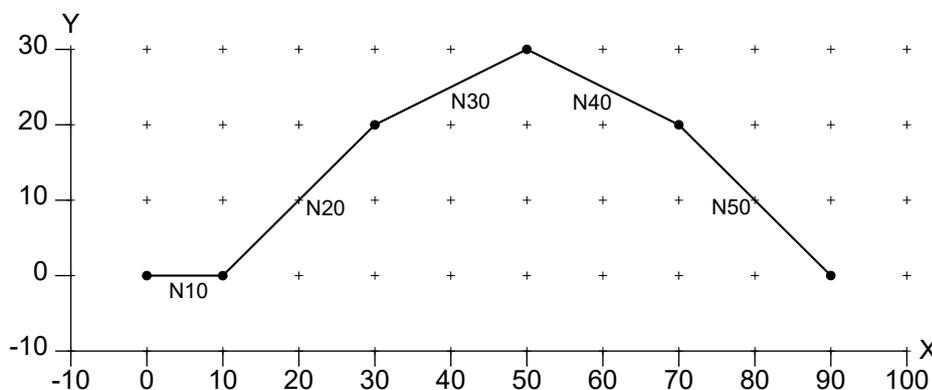


Figure "ExampleTransPop".

transDepth

transDepth() : UInt

Yields the depth of the stack of transformations, i.e. the number of active transformations. See transRestore(...), chapter [transformations \[► 95\]](#), for more details.

transRestore

transRestore(depth:= UInt)

Reduces the stack of transformations to the given depth. This command is typically used in conjunction with transDepth() to restore an earlier state of the stack.

i The current depth of the stack must not be smaller than the given depth.

Example:

The resulting path of the following example is shown in Figure “ExampleTransDepthTransRestore”. A translation to [40, 10, 0] is initially pushed onto the transformation stack. The resulting depth is stored in variable savedDepth. The following code repeatedly performs a linear move to X20 Y0 and a rotation by 45 degree. This resulting path is one half of an octagon, composed of segments N10 to N50. When N50 is processed, the transformation stack contains the initial translation and 4 rotations by 45 degree. The invocation of transRestore(savedDepth) restores the stack depth of 1 by removing all rotations. Hence, only the translation is applied to N60.

```
!VAR savedDepth : UInt; END_VAR
!transTranslate(40,10,0);
!savedDepth := transDepth();

N10 G01 X20 Y0 F6000
!transRotZ(45);
N20 G01 X20 Y0
!transRotZ(45);
N30 G01 X20 Y0
!transRotZ(45);
N40 G01 X20 Y0
!transRotZ(45);
N50 G01 X20 Y0
!transRestore(savedDepth);
N60 G01 X10 Y0
M02
```

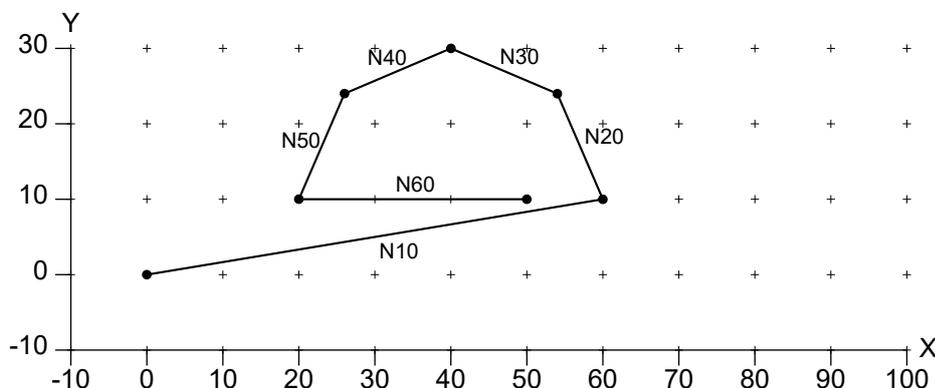


Figure “ExampleTransDepthTransRestore”.

4.6.3 Circular Movement

moveCircle3d

moveCircle3d(cx:=LReal, cy:=LReal, cz:=LReal, nx:=LReal, ny:=LReal, nz:=LReal, angle:=LReal, height:=LReal)

Move circular by rotating around the center c_x, c_y, c_z and the normal vector n_x, n_y, n_z by the given angle. If height is nonzero, a helix is described. If angle is greater than a full circle, a multiturn circle or a multiturn helix is described. The rotation is performed according to the right hand rule. Using a negative angle or flipping the normal will reverse the direction of rotation. The angle value is interpreted using the current angle unit. The parameters x, y, z, c_x, c_y, c_z are interpreted using the current length unit.

i The radius must be nonzero.

Example:

The resulting path of the following example is shown in Figure “ExampleMoveCircle3D”. The invocation of moveCircle3D describes a helical movement. It starts at the current point that is $[40, 10, 0]$. The center axis of the helix is defined by the point $[30, 10, 0]$ and direction $[gSin(22.5), 0, gCos(22.5)]$. Compared to the workingplane normal $[0, 0, 1]$ the axis is tilted by 22.5 degree in X-direction. The angle of $720+90$ degree describes a multiturn helix. It exhibits a height of 30 with respect to the center axis. The endpoint of the helix is not explicitly programmed to avoid redundancy. If the user requires these coordinates, they can be retrieved by the frameGet (...) function, as demonstrated. The approximate coordinates are shown in Figure “ExampleMoveCircle3D”.

```
{
VAR
  x, y, z: LREAL;
END_VAR

!N10 G01 X40 Y10 F6000
moveCircle3D(cx:=30, cy:=10, cz:=0, nx:=gSin(22.5), ny:=0, nz:=gCos(22.5), angle:=720+90, height:=30);
frameGet(x=>x, y=>y, z=>z);
}
M02
```

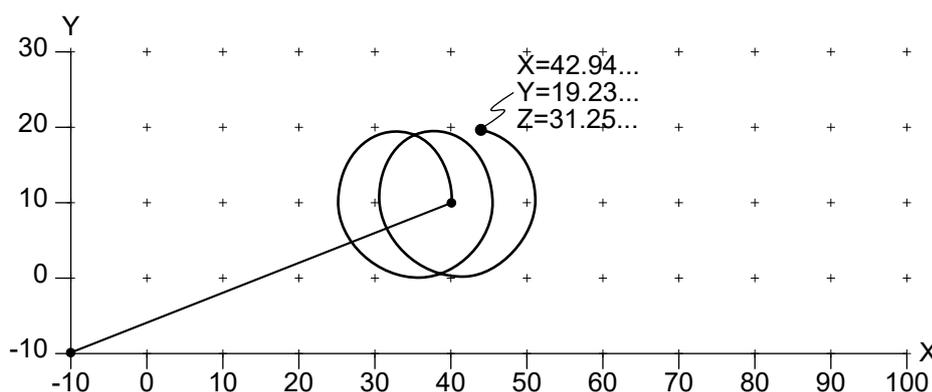


Figure “ExampleMoveCircle3D”.

4.6.4 Centerpoint Correction

centerpointCorrectionSet

centerpointCorrectionSet(on:= bool)

Activates the centerpoint correction for circles. The centerpoint correction will be used for circles that are defined using centerpoint programming. (See G2 and G3, chapter [codes \[▶ 48\]](#).) Due to inaccuracies (e.g. rounding errors by the CAD program), the radius of the starting-point and endpoint with respect to the centerpoint may differ. If centerpoint correction is active, the center will be moved in such a way that the starting-radius and endradius are equal to their former average.

A limit for centerpoint correction can be configured with `centerpointCorrectionLimitSet(...)`. If this limit is exceeded, a runtime error will be reported.

centerpointCorrectionLimitSet

```
centerpointCorrectionLimitSet(limit:= LReal)
```

Sets the precision limit for the centerpoint of circles. If the given limit is exceeded, a runtime error is reported. The default limit value is 0.1 mm.

4.6.5 Tools

toolParamSet

```
toolParamSet(tidx:= USInt, col:= USInt, val:= LReal)
```

Set a parameter of the tool `tidx` (1..255) to `val`. The parameter is identified by `col` (0..15).

COL	DESCRIPTION
0	tool number For giving the tool a number. Written to the T-parameter in the cyclic channel interface.
1	tool type (10: drill, 20: miller) The drill is type 10. The miller is type 20.
2	length Describes the length of e.g. the drill.
3	
4	radius
5	length (added to the length value of column 2) Describes the wear on e.g. the drill. The wear has to be given as a negative value as it is added to the length.
6	
7	radius (added to the radius value of column 4)
8	x-shift Cartesian tool displacement in x-direction.
9	y-shift Cartesian tool displacement in y-direction.

10

z-shift
 Cartesian tool displacement in z-direction.

toolParam

toolParam(tidix:= USInt, col:= USInt): LReal

Yields the given tool parameter.

toolSet

toolSet(index:= USInt, nr:= Int, tooltype:= ToolType, length:= LReal, radius:= LReal, lengthAdd:= LReal, radiusAdd:= LReal, offsetX:= LReal, offsetY:= LReal, offsetZ:= LReal)

Set all parameters of a tool. The index is used in D-words to refer to the tool. It must lie in the range 1 to 255. The parameter nr has only informational purpose. Typically, it is a company internal number to identify a certain tool. The parameter tooltype identifies the kind of tool, like a drill for instance. The remaining parameters are dimensions, which are visualized in Figure “ToolSetDimensions”. If the tool orientation is changed towards the negative (see P-word, chapter codes [► 48]), the value length+lengthAdd is implicitly negated. The parameters length, radius, lengthAdd, radiusAdd, offsetX, offsetY and offsetZ are interpreted using the current length unit.

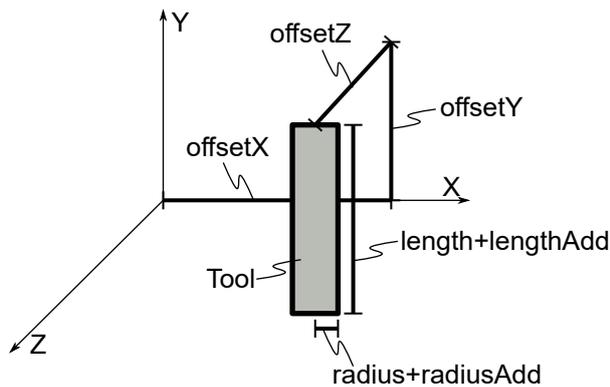


Figure “ToolSetDimensions”.

Example:

The example defines tool 1 as a drill of total length 48.5 and tool 2 as a mill with a length of 30 and a diameter of 5.

```
!toolSet(index:=1, nr:=4711, tooltype:=tooltypeDrill, length:=50, lengthAdd:=-1.5);
!toolSet(index:=2, nr:=10783, tooltype:=tooltypeMill, length:=30, radius:=2.5);
```

ToolType

Enumeration of tool types.

```
tooltypeDrill
tooltypeMill
```

tooltypeDrill: Selects a drill as a tool.
 tooltypeMill: Selects a mill as a tool.

4.6.6 Synchronization

sync

`sync()`

Synchronizes the interpreter with the associated NC-channel. The `sync()`-command blocks until all pending NC-commands are completed, i.e. until the job-queue of the NC-channel is empty. This command replaces the former `@714`-command. Oftentimes, the `sync()`-command is combined with a preceding M-function of type handshake. Then, the `sync()`-command will block until the M-function is acknowledged by the PLC.

wait

`wait()`

Waits for a GoAhead-signal from the PLC. The `wait()`-command blocks until this signal is received. This command replaces the former `@717`-command. Compared to a combination of an M-function and `sync()`, this kind of synchronization does not result in an empty job queue. Notably, an empty job queue forces the machine to halt.



The GoAhead-signal may be send from the PLC before the associated `wait()`-function is called. In this case the `wait()`-function does not block.

4.6.7 Query of Axes

queryAxes

`queryAxes()`

Set the MCS (machine coordinate system) coordinates of the interpreter to the actual coordinates of the physical axes. The MCS (machine coordinate system) coordinates are automatically translated to PCS (path coordinate system) coordinates, which are exposed to the programmer. They may also be retrieved by `frameGet(...)`. A combination of `sync()` and `queryAxes()` replaces the former `@716`-command.

- The `queryAxes()`-command considers the path axes (X, Y, Z), as well as the auxiliary axes (Q1..Q5).



The `queryAxes()`-command should be preceded by `sync()` to avoid unexpected behavior.

Example:

The resulting path of the following example is shown in Figure “ExampleQueryAxes”. The example assumes M20 to be an M-function of type “handshake after”. The PLC is assumed to

- wait for M20,
- move the Y-axis to 20,
- wait for completion of the movement,
- acknowledge M20.

The interpreter sends the line segment N10 to the NC-channel followed by the M-function M20. Then the invocation of `sync()` blocks. The NC-channel signals the M-function to the PLC after the line segment N10 has been processed. Then the PLC moves the tool from the end of segment N10 to the beginning of segment N20 and acknowledges M20. The interpreter resumes operation and invokes `queryAxes()`, which sets the internal “current point” to the endpoint of segment N10'. Therefore, the final block sends the line segment N20 to the NC-channel.

```
N00
N10 G01 X40 M20 F6000
!sync();
!queryAxes();
N20 G01 X80
M02
```

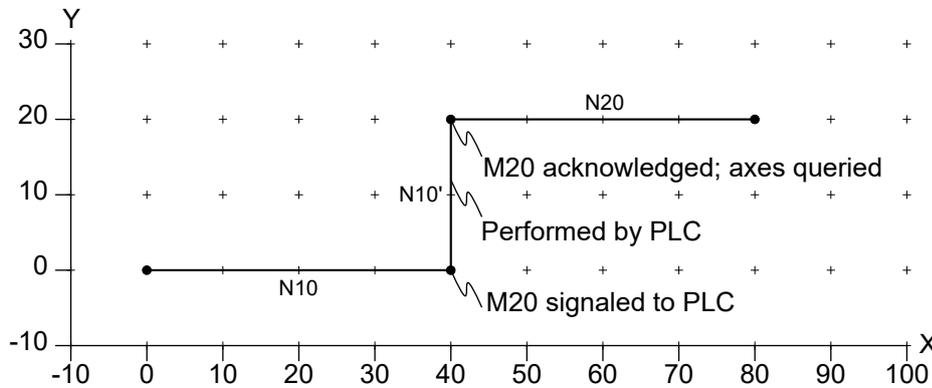


Figure "ExampleQueryAxes".

4.6.8 Current Point

frameGet

```
frameGet(x:=LReal, y:=LReal, z:=LReal, a:=LReal, b:=LReal, c:=LReal)
```

Store the current frame of the PCS (program coordinate system) in x, y, z and a, b, c.

Example:

The output of the following example is shown below. The G-Code in the example performs a linear movement to the PCS (program coordinate system) point [10, 20, 30]. Then these coordinates are stored in curX, curY, curZ by frameGet(...). The translation [1, 2, 3] that is pushed onto the transformation-stack leads to an adaption of the current PCS (program coordinate system) point such that the MCS (machine coordinate system) point [10, 20, 30] remains unchanged. Therefore, the subsequent call of frameGet(...) retrieves the PCS (program coordinate system) point [9, 18, 27].

```
{
VAR
  curX, curY, curZ : LREAL;
END_VAR

!G01 X10 Y20 Z30 F65000

frameGet(x=>curX, y=>curY, z=>curZ);
MSG(toString(curX, ' ', curY, ' ', curZ, ''));

transTranslate(1,2,3);
frameGet(x=>curX, y=>curY, z=>curZ);
MSG(toString(curX, ' ', curY, ' ', curZ, ''));
}
M02
```

Output:

```
10.000000 20.000000 30.000000
9.000000 18.000000 27.000000
```

qAxisGet

```
qAxisGet(q1:=LReal, q2:=LReal, q3:=LReal, q4:=LReal, q5:=LReal)
```

Store the current values of Q-axes in q_1 to q_5 . The Q-axes are the auxiliary axes.

4.6.9 Tool Radius Compensation

trcApproachDepartSet

```
trcApproachDepartSet(approachRadius:= LReal, approachAngle:= LReal, departRadius:= LReal, departAngle:= LReal)
```

Configures the approach and depart behavior to use an arc of given radius and angle. If the product of radius and angle are zero, no approach or depart segment will be inserted.

The resulting configuration is used by G41/G42.

trcOffsetSet

```
trcOffsetSet(offset:= LReal)
```

Configures the amount of segment extension that is used to close gaps. If *offset* is zero, no extension will be performed.

The resulting configuration is used by G41/G42.

trcLimitSet

```
trcLimitSet(limit:= LReal)
```

Configures the lookahead that is used for collision elimination.

The resulting configuration is used by G41/G42.

trcParam

```
trcParam(): TrcParamType
```

Returns the current configuration as a structure value.

trcParamSet

```
trcParamSet(param:= TrcParamType)
```

Configures the tool radius compensation. This function is an alternative that summarizes `trcApproachDepartSet`, `trcOffsetSet` and `trcLimitSet`. It can be used in combination with `trcParam` to load, save and restore different TRC (tool radius compensation) configurations efficiently.

TrcParamType

```
TrcParamType
```

This structure contains all configuration parameters of the tool radius compensation. It consists of the following parameters.

```
approachRadius: LREAL;  
approachAngle: LREAL;  
departRadius: LREAL;  
departAngle: LREAL;  
offset: LREAL;  
limit: ULINT;
```

See `trcApproachDepartSet`, `trcOffsetSet`, `trcLimitSet` for a comprehensive description of the listed parameters.

collisionElimination

```
collisionElimination(nx:= LReal, ny:= LReal, nz:= LReal, limit:= ULInt)
```

Activates collision elimination with respect to the plane of the normal vector nx , ny , nz . Collisions within the projection of the path onto the plane are eliminated. Supplying a zero vector deactivates collision elimination. The `limit` parameter can be used to restrict elimination to the last n segments. By default, elimination is unlimited.

collisionEliminationFlush

```
collisionEliminationFlush()
```

This function can be called during active collision elimination to ignore any conflicts between the path preceding the call and the path succeeding the call.

4.6.10 Suppression of G-Code Blocks

disableMask

```
disableMask(): LWord
```

Yields the current value of the disable mask. Note that the mask may also be set by the PLC.

disableMaskSet

```
disableMaskSet(mask:= LWord)
```

Sets the internal disable mask to the given value. The mask is used to suppress execution of G-Code blocks. The disable mask has 0 default value, i.e. no suppression is active by default. The mask consists of 64 bits.

In a binary notation like `2#1101` bits are numbered from right to left, starting with bit 0. For the value `2#1101` the bits 0, 2 and 3 are set by value `one`. The remaining bits are not set by exhibiting `zero` value.

Example:

The resulting path of the following example is shown in Figure “ExampleDisableMaskSet”. The disable mask is initially set to the binary value `2#1101`, which is equal to the decimal value 13. The first G-Code, which is `N10` in the given example, is always executed, independently of the current disable mask because there is no `’/’`-operator in the `N10`-line. `N20` is only executed if bit 0 is not set. In the case bit 0 is set `N20` is suppressed, which is true in the given example. The same holds for `N30`, since `’/’` is only a shorthand for `’/0’`. `N40` is not suppressed, since bit 1 is not set. The G-Codes `N50` and `N60` after `disableMaskSet(0)` are executed, since no bit in the disable mask is set. In contrast, the call `disableMaskSet(-1)` sets all bits of the mask. Consequently, the succeeding G-Codes that are prefixed with a `’/’`, `N80` and `N90`, are disabled.

```
!disableMaskSet(2#1101);
N10 G01 X10 Y10 F6000
/0 N20 G01 X20 Y0
/ N30 G01 X30 Y0
/1 N40 G01 X40 Y10
!disableMaskSet(0);
/ N50 G01 X50 Y0
/1 N60 G01 X60 Y10
!disableMaskSet(-1);
N70 G01 X70 Y0
/1 N80 G01 X80 Y10
/2 N90 G01 X90 Y20
M02
```

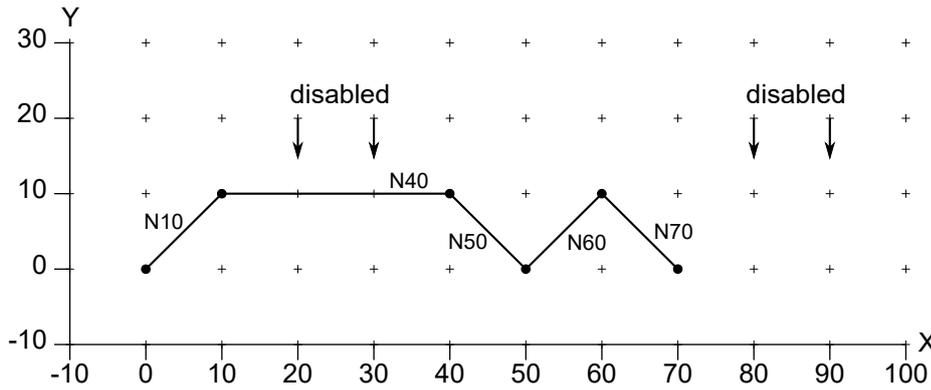


Figure "ExampleDisableMaskSet".

4.6.11 Zero Offset Shift

zeroOffsetShiftSet

zeroOffsetShiftSet(g:= USInt, x:= LReal, y:= LReal, z:= LReal)

Sets the translation for G-Code g where g must be one of the numbers 54, 55, 56 or 57. Alternatively, the Zero Offset Shift can be set with the PLC Function Block [ItpWriteZeroShiftEx](#) [▶ 247].

Example:

The resulting path of the following example is shown in Figure "ExampleZeroOffsetShiftSet". The zero offset shift of G54 is first set to the translation [0, 10, 0]. It gets active for N20 and any later segment endpoints until a novel translation is applied. The second invocation of zeroOffsetShiftSet has an immediate effect. It applies to N30 and any later segment endpoints until a novel translation is applied. The same holds for the last invocation. However, the block N40 does not program the Y-coordinate. Therefore, the change does not become apparent for N40. (See section [Transformations](#) [▶ 95] for details.) Because the block N50 programs the Y-coordinate, it applies the recent [0, 30, 0]-translation.

```
!zeroOffsetShiftSet(g:=54, x:=0, y:=10, z:=0);
N10 G01 X20 Y0 F6000
N20 G01 G54 X40 Y0
!zeroOffsetShiftSet(g:=54, x:=0, y:=20, z:=0);
N30 G01 X60 Y0
!zeroOffsetShiftSet(g:=54, x:=0, y:=30, z:=0);
N40 G01 X80
N50 G01 X100 Y0
M02
```

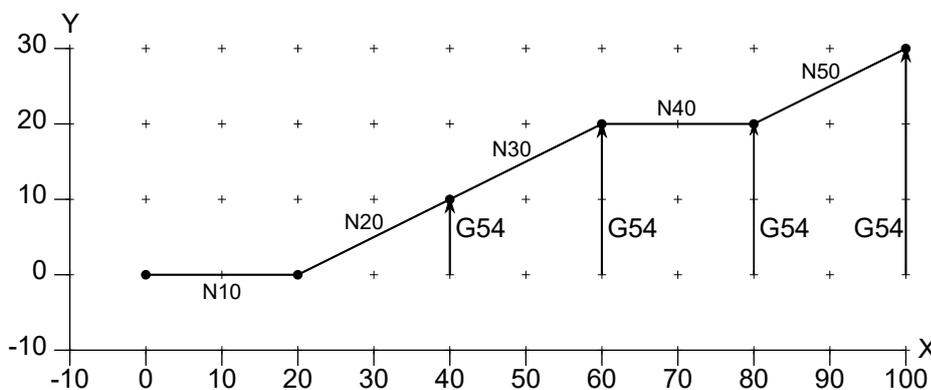


Figure "ExampleZeroOffsetShiftSet".

4.6.12 Units

unitAngleSet

```
unitAngleSet(unitAngle:= UnitAngle)
```

Set the unit for angles to `unitAngle`. The default is `unitAngleDegree`. The unit for angles applies to all NC-related functions like `transRotX`. It does not apply to ST-standard functions like `sin`. For this reason the interpreter offers a set of NC-specific counterparts like `gSin` that consider the angle unit.

UnitAngle

Enumeration of the following values:

```
unitAngleRadian: 0...2pi  
unitAngleDegree: 0...360  
unitAngleGon: 0...400  
unitAngleTurn: 0...1
```

unitLengthSet

```
unitLengthSet(unitLength:= UnitLength)
```

Set the unit for lengths to `unitLength`. The default is `unitLengthMillimeter`. The unit for length applies to all NC-related functions like `G01` or `zeroOffsetShiftSet(...)`.

UnitLength

Enumeration of the following values:

```
unitLengthMeter  
unitLengthCentimeter  
unitLengthMillimeter  
unitLengthMicrometer  
unitLengthNanometer  
unitLengthInch  
unitLengthFoot
```

unitTimeSet

```
unitTimeSet(unitTime:= UnitTime)
```

Set the unit for time to `unitTime`. The default is `unitTimeSecond`. The unit for time applies to all NC-related functions like `G04`. It does not apply to ST-standard functions like `currentLdt()`.

UnitTime

Enumeration of the following values:

```
unitTimeSecond  
unitTimeMillisecond  
unitTimeMicrosecond  
unitTimeMinute  
unitTimeHour
```

unitVelocitySet

```
unitVelocitySet(unitLength:= UnitLength, unitTime:= UnitTime)
```

Set the unit for velocity to `unitLength/unitTime`. The default is `unitLengthMillimeter/unitTimeMinute`. The unit for velocity applies to all NC-related functions. It is used by the `F`-parameter for instance.

4.6.13 Trigonometric (Unit Aware)

gSin

`gSin(angle:= LReal)`

Returns the sine of the given `angle` where the current angle unit is used to interpret the angle. (See section [Units \[► 83\]](#) for details.) The return type is `LReal`. This function is not overloaded.

gCos

`gCos(angle:= LReal)`

Returns the cosine of the given `angle` where the current angle unit is used to interpret the angle. (See section [Units \[► 83\]](#) for details.) The return type is `LReal`. This function is not overloaded.

gTan

`gTan(angle:= LReal)`

Returns the tangent of the given `angle` where the current angle unit is used to interpret the angle. (See section [Units \[► 83\]](#) for details.) The return type is `LReal`. This function is not overloaded.

gASin

`gASin(val:= LReal)`

- Returns the arcsine of `val` in the current angle unit. (See section [Units \[► 83\]](#) for details.)
- The return type is `LReal`. This function is not overloaded.
- The result lies within the interval $[-c/4, c/4]$ where c is the angle of a full circle in the current angle unit.

● CONSTRAINT:

i The variable `val` must reside within the interval $[-1, 1]$.

gACos

`gACos(val:= LReal)`

- Returns the arccosine of `val` in the current angle unit. (See section [Units \[► 83\]](#) for details.)
- The return type is `LReal`. This function is not overloaded.
- The result lies within the interval $[0, c/2]$ where c is the angle of a full circle in the current angle unit.

● CONSTRAINT:

i The variable `val` must reside within the interval $[-1, 1]$.

gATan

`gATan(val:= LReal)`

- Returns the arctangent of `val` in the current angle unit. (See section [Units \[► 83\]](#) for details.)
- The return type is `LReal`. This function is not overloaded.

- The result lies within the interval $[-c/4, c/4]$ where c is the angle of a full circle in the current angle unit.

gATan2

`gATan2 (y:= LReal, x:= LReal)`

- Returns the arctangent of y/x in the current angle unit. (See section [Units](#) [▶ 83] for details.)
- The return type is `LReal`. This function is not overloaded.
- The result lies within the interval $[-c/2, c/2]$ where c is the angle of a full circle in the current angle unit.

4.6.14 Feed Mode

feedModeSet

`feedModeSet (feedMode:= FeedModeType)`

FeedModeType

Enumeration of the following values:

```
fmContour  
fmInternalRadius  
fmToolCenterPoint
```

fmContour: Holds the feedrate at the contour constant.

fmInternalRadius: Reduces the feedrate at internal radii. This results in a constant velocity at the contour. The velocity at external radii is not increased.

fmToolCenterPoint: Keeps the feedrate of the tool's center point constant. This means that at internal radii the velocity at the contour is increased, and that it is correspondingly reduced at external radii.

4.6.15 Feed Interpolation

feedInterpolationSet

`feedInterpolationSet (feedInterpolation:= FeedInterpolationType)`

FeedInterpolationType

`FeedInterpolationType`

Enumeration of the following values:

```
fiConstant  
fiLinear
```

fiConstant: The programmed velocity is applied as fast as possible with the constant feed interpolation (default).

fiLinear: The linear feed interpolation transfers the velocity linearly over the path from v_start to v_end .

4.6.16 Streaming of Large G-Code Files

runFile

```
runFile(path:= string)
```

The size of files that can be executed employing the GST-interpreter is limited. However, sometimes it is required to execute large files that may have been created e.g. by a CAD-program. Therefore, the user has the possibility to execute filestreams of native G-Code.

Executes the plain G-Code that is contained in the G-Code file given by `path`. The function call returns after all lines in the supplied file have been processed. The function is intended for streaming large G-Code files to the NC-kernel efficiently.

i Native G-Code: No Structured Text Allowed

Note that the supplied G-Code file must not contain any ST-elements, but only plain G-Code.

A G-Code filestream from file 'myNativeGCodeFile.nc' can be called from a GST-program lining up the following syntax:

```
!runfile('myNativeGCodeFile.nc');
```

4.6.17 Vertex Smoothing

smoothingSet

```
smoothingSet(mainType:= SmoothingMainType, subType:= SmoothingSubType, value:= L
Real)
```

Sets the vertex smoothing behavior according to the given parameters.

SmoothingMainType

Enumeration of the following values:

```
smoothingNone
smoothingParabola
smoothingBiquadratic
smoothingBezier3
smoothingBezier5
smoothingTwinBezier
```

smoothingNone: No smoothing.

smoothingParabola: For parabola smoothing a parabola is inserted geometrically into the segment transition. This ensures a steady velocity transition within the tolerance radius.

smoothingBiquadratic: With biquadratic smoothing there is no step change in acceleration in the axis components. With the same radius, a smaller input velocity may therefore be required than for parabolic smoothing.

smoothingBezier3: In case of the 3rd order Bezier curve a step change in acceleration appears in the axis components when the tolerance sphere is entered. The max. size is limited by the acceleration of the axis components and the C1 factor.

smoothingBezier5: With 5th order Bezier blending, no step change in acceleration occurs in the axis components on entry into the tolerance sphere. In other words, the path axis acceleration is always constant if blending is selected.

smoothingTwinBezier: With the aid of smoothing, it is possible to insert a Bezier spline automatically between two geometrical entries. It is only necessary to program the radius of the tolerance sphere. This describes the maximum permissible deviation from the programmed contour in the segment transition. The advantage of this type of smoothing as opposed to rounding with an arc is that there are no step changes in acceleration at the segment transitions.

● Acute angles at the segment transition

i The Bezier splines are generated by default, even at very acute angles. In order to avoid the dynamic values being exceeded, a considerable reduction velocity is required in this case. However, since the dynamics are held constant in the spline, the movement across the spline can be quite slow. In this case it is often practical to start the segment transition with an accurate stop. The command [AutoAccurateStop](#) [► 155] can be used to avoid having to calculate the angles manually.

SmoothingSubType

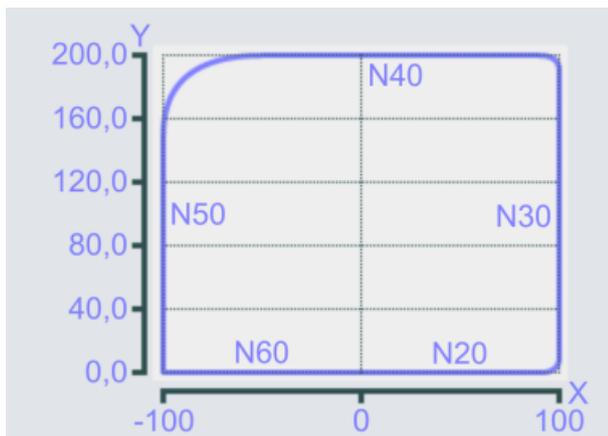
Enumeration of the following values:

```
smoothingRadius
smoothingDistance
smoothingAdaptive
```

Example

The example visualizes the effect of using a smoothing parabola. In the first two corners smoothing value 10 and in the third corner smoothing value 50 have been used. Finally, the fourth corner exhibits smoothing value 0.

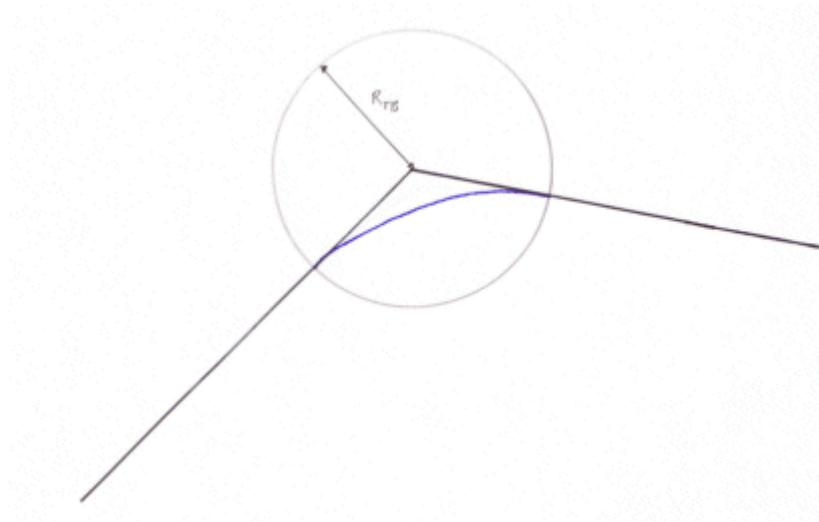
```
N10 G01 X0 Y0 F60000
!smoothingSet(mainType:=smoothingParabola, subType:=smoothingRadius, value:=10);
N20 G01 X100
N30 Y200
!smoothingSet(mainType:=smoothingParabola, subType:=smoothingRadius, value:=50);
N40 X-100
!smoothingSet(mainType:=smoothingParabola, subType:=smoothingRadius, value:=0);
N50 Y0
N60 X0
M02
```



4.6.17.1 Subtypes

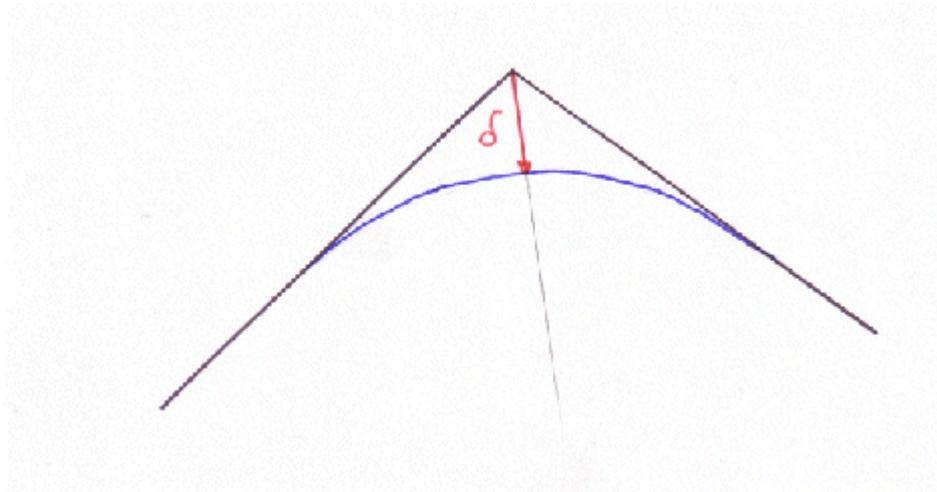
SmoothingRadius (subtype 1)

If subtype 1 is selected, the maximum tolerance radius (R_{TB}) is used for blending. R_{TB} is reduced if and only if the input or output segment is less than $3 \cdot R_{TB}$.



SmoothingDistance (subtype 2)

The distance between the programmed segment transition and the vertex of the parabola is specified with the subtype 2. The tolerance radius (R_{TB}) results from this. If a segment is too short, then the distance is shortened so that the tolerance radius is a max. of 1/3.



SmoothingAdaptive (subtype 3)

Within the tolerance radius (including constant tolerance radius) the system ensures that the maximum permissible acceleration is not exceeded. Depending on the deflection angle and the velocity, the maximum axis acceleration within the smoothing segment may be different. The aim of an adaptive tolerance radius is maximum acceleration during smoothing. In order to achieve this, the smoothing radius is reduced based on the programmed velocity and dynamics. In other words, if the programmed velocity is changed, the tolerance radius can also change. The override has no influence on the radius.

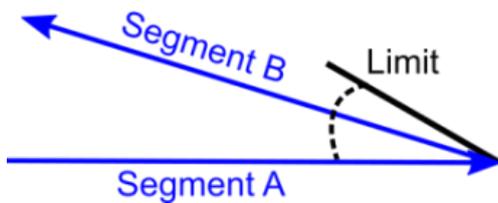
4.6.18 Automatic Accurate Stop

autoAccurateStopSet

```
autoAccurateStopSet (angle:= LREAL);
```

The command `autoAccurateStopSet` is used in conjunction with `blending` (see [smoothingSet](#)) and allows driving to acute angles with active blending. A limit angle, up to which an accurate stop between 2 segments must take place, is defined for this in the command `autoAccurateStopSet`.

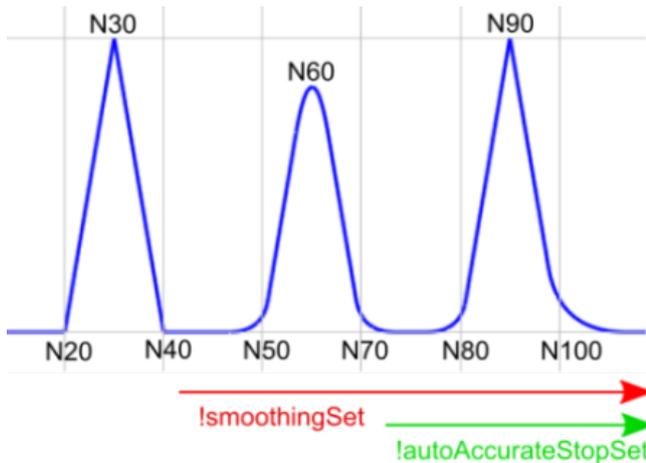
```
!autoAccurateStopSet (angle:= 30.0);
```



For circle segments, the angle is calculated from the tangents at the points of entry and leaving.

Sample

```
N10 G0 X0 Y0 Z0
N20 G01 X10 F20000
N30 G01 X15 Y30
N40 G01 X20 Y0
!smoothingSet(mainType:=smoothingParabola,
subType:=smoothingRadius, value:=50);
N50 G01 X30
N60 G01 X35 Y30
N70 G01 X40 Y0
!autoAccurateStopSet(angle:= 46.0);
N80 G01 X50
N90 G01 X55 Y30
N100 G01 X60 Y0
N110 G01 X80
N110 M30
```



Requirements

Development environment	Target system
TwinCAT V3.1.4024.15	PC or CX (x86 or x64)

4.6.19 Spline Interpolation

transBSpline

```
transBSpline (BreakAngle:=LReal, BreakLength:=LReal, MergeDiff:=LReal,
LineBreakAngle:=LReal,LineBreakLength:=LReal, LineMergeDiff:=LReal)
```

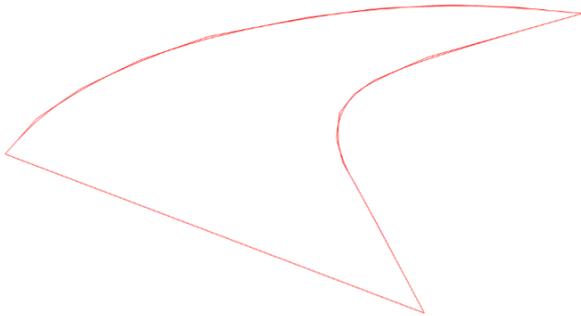
transBSpline generates a continuous curve from a piecewise linear polyline.

The curve is bounded by the input polyline, the start and end points are interpolated, interior points are the control points (DeBoor points) of the curve. At least three points are required. A BSpline curve exhibits local control and is thereby amenable to control point manipulation.

```
//Enable
transBSpline(BreakAngle := 70, BreakLength := 1000);
```

```
//Disable
transBSpline();
```

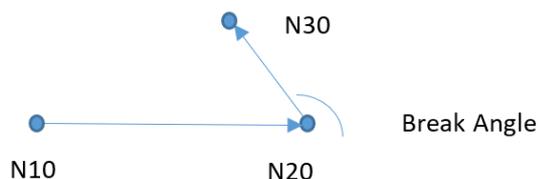
Example:



```
!//BSpline
N10 G00 X18.498 Y0
!transBSpline(BreakAngle:=70.0, BreakLength:=1000.0);
N20 G01 X18.498 Y0 Z0 F6000
N30 X16.572 Y6.543 Z1
N40 X15.616 Y9.715 Z2
N50 X15.121 Y11.275 Z3
N60 X14.838 Y13.196 Z4
N70 X14.982 Y15.085 Z5
N80 X15.595 Y16.485 Z6
N90 X16.396 Y17.490 Z7
N100 X18.653 Y19.243 Z8
N110 X25.07 Y22.526 Z9
N120 X22.228 Y22.997 Z8
N130 X19.569 Y23.174 Z7
N140 X16.488 Y22.884 Z6
N150 X13.634 Y22.228 Z5
N160 X9.533 Y20.793 Z4
N170 X6.668 Y19.009 Z3
N180 X4.224 Y16.877 Z2
N190 X2.376 Y14.61 Z1
N200 X1.068 Y11.959 Z0
! transBSpline();
M02
```

Parameters

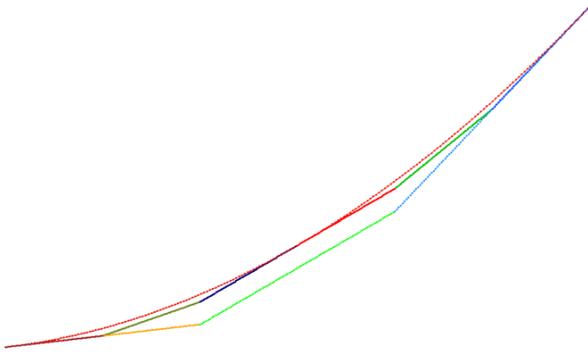
BreakAngle (mandatory): Allows preservation of sharp angle features in the path. The spline will break when the path deviates more than BreakAngle. The spline will terminate and interpolate the point.



BreakLength (mandatory): Allows preservation of long features in the path. The spline will break for segments longer than BreakLength. The spline will terminate and interpolate the start and end points of the long segment.



MergeDiff (optional): The BSpline is comprised of Bezier segments. To potentially improve processing speed the spline may be compressed by merging. Adjacent segments will be merged together when the difference in control points is less than MergeDiff. Below adjacent segments are merged into one.



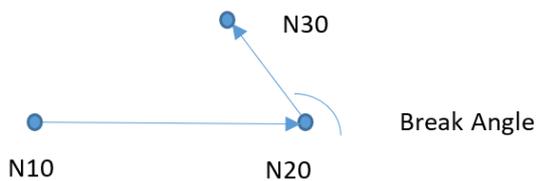
i Excessive curvature

Overly aggressive merging can result in excessive contortion and a segment of excessive curvature will be rejected with a run time error.

Acceptable curvature is derived from path velocity and acceleration.

The BSpline is constructed from a control point polyline formed by G01 segments, eg: CAD/CAM. To improve processing speed the control point polyline may be compressed or simplified by merging adjacent segments.

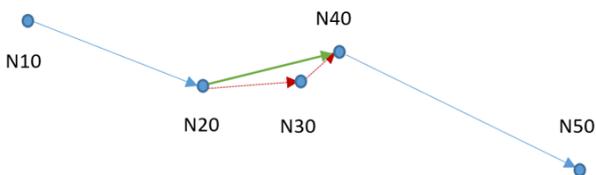
LineBreakAngle (optional): Merging of adjacent control points will break if the deviation angle exceeds LineBreakAngle.



LineBreakLength (optional): Merging of adjacent control points will break if the length exceeds LineBreakLength.



LineMergeDiff (optional): Adjacent control points will be merged if the difference (perpendicular distance) is less than LineMergeDiff. In the example N30 may be eliminated, simplifying the control polygon.



i If the optional parameters aren't parameterized or if they are 0, no merging will take place.

i Excessive curvature

Overly aggressive merging can result in excessive contortion and a segment of excessive curvature will be rejected with a run time error.

Acceptable curvature is derived from path velocity and acceleration.

Processing order:

If the BreakAngle or BreakLength are 0. No further processing will take place. LineBreakAngle, LineBreakLength and LineMergeDiff are processed firstly to simplify the control point polyline. BreakAngle BreakLength and MergeDiff are processed finally to generate the BSpline curve.

Decoder Stops and Handshake M functions:

The BSpline should be terminated with `!transBSpline()`; prior to either a decoder stop or a M-function type handshake.

```
!//BSpline
N10 G00 X18.498 Y0
!transBSpline(BreakAngle:=70.0, BreakLength:=1000.0);
N20 G01 X18.498 Y0 Z0 F6000
N30 X16.572 Y6.543 Z1
N40 X15.616 Y9.715 Z2
N50 X15.121 Y11.275 Z3
N60 X14.838 Y13.196 Z4
N70 X14.982 Y15.085 Z5
N80 X15.595 Y16.485 Z6
N90 X16.396 Y17.490 Z7
N100 X18.653 Y19.243 Z8
N110 X25.07 Y22.526 Z9
!transBSpline();
!sync();
!transBSpline(BreakAngle:=70.0, BreakLength:=1000.0);
N120 X22.228 Y22.997 Z8
N130 X19.569 Y23.174 Z7
N140 X16.488 Y22.884 Z6
N150 X13.634 Y22.228 Z5
N160 X9.533 Y20.793 Z4
N170 X6.668 Y19.009 Z3
N180 X4.224 Y16.877 Z2
N190 X2.376 Y14.61 Z1
N200 X1.068 Y11.959 Z0
!transBSpline();
M02
```

Compatible G-Codes and functions

G-Codes other than G01 are supported.

G00

G02, G03 (Circle and Helix): The BSpline will terminate before and continue afterwards.

G04

G09, G60

G54 and other transformations

```
disableMask()
runFile(path:= )
smoothingSet(mainType:=smoothingTwinBezier,subType:=smoothingRadius,value:= )
```

ToolRadiusCompensation is not supported.

Requirements

Development Environment	Target System
TwinCAT V3.1.4024.4	PC or CX (x86 or x64)

4.6.20 Dynamic Override**dynOverrideSet**

`dynOverrideSet(value:= LReal)`

Set the dynamic override of axes to the given `value`.

The dynamic override function can be used to implement and evoke percentage changes to the dynamic axis parameters in the axis group while the NC program is running. Thus, these changes result in new values for motion dynamics. Without any stop the new dynamic values become valid when the line is executed.

Range of Values

The factor `value` for `dynOverrideSet` has to reside within the range $0 < \text{value} \leq 1.0$.

Example

Within the example the new dynamic values become valid without any stop. In block N010 the previously set values are used for deceleration, while the changed values are used for acceleration in block N020.

```
N010 G01 X100 Y200 F6000
!dynOverrideSet(value:= 0.4);
N020 G01 X500
M02
```

Example

The command `dynOverrideSet` can be used to reduce acceleration and jerk e.g. only for one movement. In the example acceleration and jerk are reduced by 50 percent merely in block N020.

```
N010 G01 X100 Y100 F6000
!dynOverrideSet(value:= 0.5);
N020 X0
!dynOverrideSet(value:= 1);
N030 X100
M02
```

4.6.21 Center Point Reference of Circles

circleCenterReferenceSet

```
circleCenterReferenceSet(value:= ReferenceType)
```

- Sets the center reference type for circles that are programmed by G02/G03 using a center point, whose definition involves the `i,j,k`-parameters.
- For `referenceAbsolute` the center point of the circle is defined by the supplied `i,j,k`-vector.
- For `referenceRelative` the center point is defined by the sum of the circle starting-point and the supplied `i,j,k`-vector. This is the default and usual behavior of G-Code.

ReferenceType

Enumeration of the following values:

```
referenceAbsolute
referenceRelative
```

4.6.22 Change in axis dynamics

axisDynamicsSet

```
axisDynamicsSet(axisNo:= UDIInt, acc:= LReal, dec:= LReal, jerk:= LReal);
```

`axisDynamicsSet` can be used to change the axis dynamics at runtime.

Function	<code>axisDynamicsSet</code>
----------	------------------------------

Parameter <axisNo>	Axis in the interpolation group: X: 0 Y: 1 Z: 2 Q1: 3 ... Q5: 7
Parameter <acc>	Value of the maximum permitted acceleration in mm/s ² .
Parameter <dec>	Value of the maximum permitted deceleration in mm/s ² .
Parameter <jerk>	Value of the maximum permitted jerk in mm/s ³ .

Example:

```
N10 G01 X100 Y200 F6000
!R4:=10000;
!axisDynamicsSet(axisNo:= 0, acc:= 2250, dec:= 2250, jerk:= R4);
N30 G01 X500
N40 M02
```

Requirements

Development Environment	Target System
TwinCAT V3.1.4024.4	PC or CX (x86 or x64)

4.6.23 Change in path dynamics**pathDynamicsSet**

```
pathDynamicsSet(acc:= LReal, dec:= LReal, jerk:= LReal);
```

pathDynamicsSet can be used to change the path dynamics at runtime.

Function	pathDynamicsSet
Parameter <acc>	Value of the maximum permitted acceleration in mm/s ² . Must be set >= 1. If set to 0, the default value is used.
Parameter <dec>	Value of the maximum permitted deceleration in mm/s ² . Must be set >= 1. If set to 0, the default value is used.
Parameter <jerk>	Value of the maximum permitted jerk in mm/s ³ . Must be set >= 1. If set to 0, the default value is used.

Example:

```
N10 G01 X100 Y200 F60000
!R4:=10000;
//Set path dynamics
!pathDynamicsSet(acc:=200, dec := 200, jerk := R4);
N30 G01 X500 Y0
//Set path dynamics back to default values and jerk to 12000
!pathDynamicsSet(acc:=0, dec := 0, jerk := 12000);
N50 G01 X100 Y200
//Set path dynamics to default values
!pathDynamicsSet(acc:=0, dec := 0, jerk := 0);
N70 G01 X500 Y0
N80 M02
```

Requirements

Development Environment	Target System
TwinCAT V3.1.4024.12	PC or CX (x86 or x64)

4.7 Transformations

Speaking of GST-Transformations we refer e.g. to rotations or to zero-point-shifts.

The relation between the MCS (machine coordinate system) and the PCS (program coordinate system) is defined by the effective transformation T . T is the concatenation of the transformations T_Z , T_U and T_T ($T = T_Z * T_U * T_T$). Note that the order of concatenation is significant for the transformations do not commute in general. The transformation T_Z represents a (historical) zero offset shift, the transformation T_U represents a user defined transformation and the transformation T_T represents a tool transformation. They are described in detail later.

Figure “TransformationsTzTuTt” visualizes the relation between the MCS (machine coordinate system) and the PCS (program coordinate system):

- T_Z is defined to be a translation by $[20, 20, 0]$,
- T_U is a combination of the translation $[30, -10, 0]$, followed by a rotation by 45 degree around the z-axis,
- T_T is a translation by $[0, -10, 0]$.

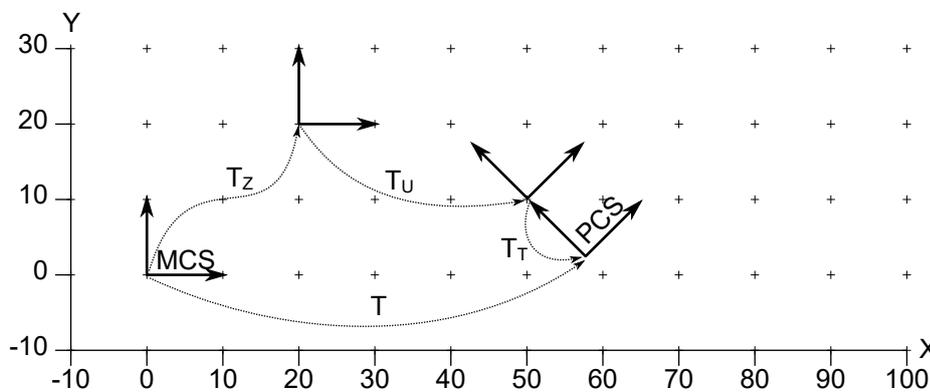


Figure “TransformationsTzTuTt”.

4.7.1 Modification of the Effective Transformation T and its Effect

Most G-Codes define only the destination point of a movement. Therefore, the interpreter maintains the current position of the tool. This point can be represented in MCS (machine coordinate system) coordinates and PCS (program coordinate system) coordinates while the equation

$CurrentPointMCS = T * CurrentPointPCS$ holds. In contrast to the previous implementation, this transformation equation also holds after a modification of T . This behavior is accomplished by adapting the $CurrentPointPCS$. The MCS (machine coordinate system) point is not adapted, as this would affect the machine. This behavior may be summarized roughly as: When the active transformation is changed, the current PCS (program coordinate system) point is adapted in a way that the modification shows no effect.

Example:

After N10 the coordinates of the current PCS (program coordinate system) and MCS (machine coordinate system) point are $[20, 10, 80]$, since no transformation is active. The translation changes the current PCS (program coordinate system) point to $[28, 7, 84]$. Applying the translation on this point yields the unchanged MCS (machine coordinate system) point $[20, 10, 80]$. Hence, the translation shows no effect, although it is active. The block N20 programs a movement to the PCS (program coordinate system) point $[25, 7, 10]$, which is mapped to the MCS (machine coordinate system) coordinate $[17, 10, 6]$. After the invocation of `transPop()` the current PCS (program coordinate system) point is set to the current MCS (machine coordinate system) point.

```
N10 G01 X20 Y10 Z80 F6000
!transTranslate(-8,3,-4);
N20 G01 X25 Z10
```

```
!transPop();
M02
```

Example:

If the user wants the PCS (program coordinate system) point to remain unchanged, he has to retrieve and program it, as shown in the following code. However, the desire for an unchanged PCS (program coordinate system) point typically indicates a bad programming style. Actually, there should be no need for the following code.

```
{
VAR
  pcsX, pcsY, pcsZ : LREAL;
END_VAR

// ... G-Code ...

frameGet(x=>pcsX,y=>pcsY,z=>pcsZ);
// ... modify transformations ...
!G01 x=pcsX y=pcsY z=pcsZ F6000
}
```

4.7.2 Components of the Effective Transformation T

Zero Offset Shift T_z

The T_z -transformation is affected by certain G-Codes. It has no effect if G53 is active. Otherwise, T_z is the combination of the three translations T_{Z58} , T_{Z59} and one of $\{T_{Z54}, \dots, T_{Z57}\}$. The former two translations are set via the G-Codes G58 and G59. The latter translation is selected by the G-Codes G54 to G57. One translation is associated with each of these 4 G-Codes. It can be set by the PLC or using the ST-function `zeroOffsetShiftSet`.

Tool Transformation T_T

T_T is defined by the currently selected tool. It has no effect if tool 0 (D0) is selected. Otherwise, it is a translation by `[offsetX, offsetY, offsetZ] + (length+lengthAdd) * D` where D is the normal of the current workingplane.

Userdefined Transformation T_U

T_U is defined by a stack of transformations. The stack of depth N contains elementary transformations $T_{U1}, T_{U2}, \dots, T_{U<N>}$ where $T_{U<N>}$ is the topmost transformation. Initially, the stack is empty. The userdefined transformation is the concatenation of these elementary transformations $T_U = T_{U1} * T_{U2} * \dots * T_{U<N>}$. Note that the order is significant for the transformations do not commute in general. If the stack is empty, T_U is the identity transformation, which has no transformation effect.

4.7.3 Applying Transformations

A transformation is pushed onto the stack by the following ST-functions. The transformation pushed recently will be the topmost transformation on the transformation stack. When a transformation is pushed onto the transformation stack, the stack depth is increased by one and T_U is adapted accordingly.

```
transTranslate(x:= LREAL, y:= LREAL, z:= LREAL);
(* A rotation pushed onto the stack of transformations is interpreted around the respective
axis using the current angle-unit, e.g. degree or radian. *)
transRotX(angle:= LREAL);
transRotY(angle:= LREAL);
transRotZ(angle:= LREAL);
transMirrorX();
transMirrorY();
transMirrorZ();
transScale(factor:= LREAL);
```

4.7.4 Revoking Transformations

The function `transPop()` removes the topmost transformation from the transformation stack. When `transPop()` removes a transformation from the transformation stack, the stack depth is reduced by one and T_U is adapted accordingly. Commonly, the `transPop()`-function is used to revoke a temporary transformation.

Example:

In the following example the translation is applied to N10, N20 and N30. The rotation is only applied to N20 as it is revoked by `transPop()`. Figure “ExampleRevokingTransformations” shows the resulting MCS (machine coordinate system) path. Note that the rotation center is $[20, 0, 0]$ in MCS (machine coordinate system), which is the origin in PCS (program coordinate system) after the preceding translation.

```
!transTranslate(20,0,0);
N10 X10 Y0 F6000
!transRotZ(90);
N20 X20 Y0
!transPop();
N30 X30 Y0
!transPop();
M02
```

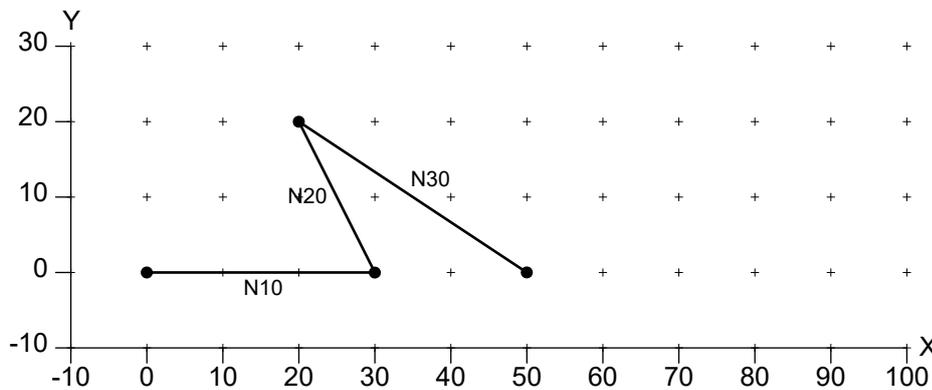


Figure “ExampleRevokingTransformations”.

4.7.5 Restoration of Stack

The function `transDepth()` yields the current depth of the stack. The function `transRestore(depth)` removes transformations from the stack until the given depth is reached. Typically, the two functions are combined to save and restore the state of the transformation stack.

It is good programming style to do this saving and restoring in the context of userdefined `ST`-functions.

Example:

Initially, within the following function the depth of the stack is stored in the variable `depth`. At the end of the function the initial state is restored by `transRestore`. Note that restoration only works properly if the stack depth does not fall below `depth` within the function. Instead of using `transDepth()` and `transRestore()` the stack depth could also be restored using `transPop()`. However, it may become cumbersome to keep pushing and popping of transformations synchronous, especially if transformations are pushed conditionally.

```
{
FUNCTION draw
VAR
    depth : UINT;
END_VAR
    depth := transDepth();
    transTranslate(10,0,0);
    // ... G-Code ...
```

```

    transRotZ(45);
    // ... G-Code ...
    transMirrorX();
    // ... G-Code ...
    transRestore(depth);
END_FUNCTION
}

```

4.8 Error Reporting

Efficient development of CNC-programs requires decent support by development tools. This support includes proper reporting of programming errors for both, compile-time errors and runtime errors. An error message should point directly to the source code the error originates from and give a precise description of the circumstances under that the error occurred (dynamic data). Such individual error messages help a developer substantially to fix errors in short time. The GST-interpreter yields such error messages, as described in the following texts.

4.8.1 Error Messages

In case of an error the interpreter produces a descriptive error message. An error message consists of a source code coordinate and a description. The source code coordinate links the error to its origin in the GST-program. It defines a range of source code stretching from the first character of the code range to the last character of the code range. Both, first and last character, are defined by their file, line and column. Note that the last character actually points to the first character behind the range, which is a common technical convention.

Example:

In the following example an integer variable `i` is declared and initialized. The initialization uses a floatingpoint literal. Since an implicit conversion from floatingpoint to integer is not allowed in ST, the interpreter produces the descriptive error message given below when the program is loaded. The error message does not only report that a type-error has occurred, it also gives the precise position: File `aaa.nc`, line 3, column 14 to 17. This code range displays the literal `'1.5'`. In addition, the programmed type (`real`) and the expected type (`int`) are reported. With such a detailed error message bugs can be fixed by the developer easily.

```

{
VAR
    i : int := 1.5;
END_VAR
}
M02

```

Error message:

```

aaa.nc: 3.14-3.17: Invalid implicit conversion from type
'<real literal>' to 'int'.

```

4.8.2 Compile-Time Errors and Runtime Errors

Errors may occur during program loading (so called compile-time errors) or during program execution (so called runtime errors). Fortunately, most errors are detected at compile-time. This detection includes missing files, syntax errors, type errors and unexpected identifiers. The developer gets feedback immediately when he tries to load the program. Thus, a part of unexpected failures during machining is avoided.

However, there are also errors that, by their nature, cannot be detected at compile-time. For instance, this circumstance includes a division by zero, since the divisor may be computed dynamically. If a runtime error occurs, the interpreter is stopped safely and an error message is produced. A runtime error message is similar to a compile-time error message. It even includes a reference to the pertinent source code.

Example:

In the following example the FOR-loop contains a division of 10 by the loop variable `i`. Since the variable `i` is iterated from `-3` to `3`, this program leads to an error during the 4th iteration, when `i` has the value `0`. This error is detected at runtime and stops the interpreter. The error message shown below is reported. It points precisely to the code `'10/i'` in the example.

FILE `aaa.nc`:

```
{
VAR
  i, j : int;
END_VAR

FOR i := -3 TO 3 DO
  j := j + 10/i;
END_FOR;
}
M02
```

Error message:

```
aaa.nc: 7.12-7.16: Division by zero
```

Example:

At runtime the interpreter also performs checking of array bounds. Consequently, invalid indices do not result in unpredictable and typically fatal crashes. The runtime error message precisely defines location and origin of the error at `'idx'`, reports the erroneously supplied index (`20`) and the valid index range (`10..19`).

FILE `aaa.nc`:

```
{
VAR
  idx : INT;
  a : ARRAY [10..19] OF INT;
END_VAR

FOR idx := 10 TO 20 DO
  a[idx] := i;
END_FOR;
}
M02
```

Error message:

```
aaa.nc: 8.5-8.8: Out of bounds. 20 exceeds range 10..19.
```

4.8.3 Errors in G-Code

Error reporting is also performed for G-Code. This reporting includes compile-time errors and runtime errors. Runtime error reporting includes invalid use of G-Code, e.g. like a bad definition for a circle.

Example:

In the following example the value of a string variable `str` is assigned to the address letter `X` of the G-Code block. As before the position of the error is precisely identified at `'=str'` in the code. In addition, the programmed type and the expected type are reported.

FILE `aaa.nc`:

```
{
VAR
  str : string := 'Hello World';
END_VAR
}

G01 X=str F6000
```

```
G01 Y100
M02
```

Error message:

```
7.5-7.9: Invalid implicit conversion from type 'string[255]'
to 'lreal'
```

Example:

In the following example a sequence of circular arcs is processed by a `FOR`-loop. The radius of the arc is 4. The distance between the starting-point and endpoint of the arc is successively increased within each iteration. During the 9th iteration the distance exceeds the circle diameter of 8. The reported error message identifies the origin `G2 X=i*10+i U4` and gives information about the diameter and distance between starting-point and endpoint.

FILE aaa.nc:

```
{
VAR
  i : INT;
END_VAR

!G00 X0 Y0 Z0
FOR i := 1 TO 10 DO
  !G01 X=i*10 F6000
  !G02 X=i*10+i U4
END_FOR;
}
M02
```

Error Message:

```
aaa.nc: 9.4-10.1: Invalid definition of circle. Distance
between start-point and end-point (=9.000000) is larger than
diameter (=8.000000).
```

4.8.4 Preprocessing

During preprocessing `#include`-directives are replaced by the contents of the referenced files. Care has been taken to maintain information about the origin of source code properly. Therefore, an error that is caused by code in an included file will refer to that included file and not to the result of preprocessing, as a simple implementation would do.

Example:

In the following example the file `aaa.nc` includes the file `bbb.nc`. In the latter file the variables `i` and `j` are used in G-Codes. Variable `i` is declared at the beginning of `aaa.nc`, but `j` is not. Therefore, the error message below is issued. As you can see it references the use of variable `j` in file `bbb.nc` properly.

FILE aaa.nc:

```
{
VAR
  i : INT;
END_VAR
}

G00 X0 Y0 Z0

#include "bbb.nc"

G00 X100

M02
```

FILE bbb.nc:

```
G01 X=i F6000
G01 Y=j
G01 Z100
```

Error message:

```
bbb.nc: 2.6-2.7: Undeclared variable or enumeration value
'j'
```

4.9 General Command Overview

Preprocessor

Com-mand	Description
<u>#include</u> [▶ 31]	The #include directive inserts the contents of another file. The included file is referenced by its path. Typically, it is used to “import” commonly used code like e.g. libraries. Its behavior is similar to the C-Preprocessor.

Interpolations

Com-mand	Description	Modal or Non-modal	Default
<u>G00</u> [▶ 37]	Interpolation mode: Linear. Applying maximum velocity ignoring programmed velocity. Reset by G01, G02, G03.	Modal.	Default.
<u>G01</u> [▶ 38]	Interpolation mode: Linear. Applying programmed velocity. Reset by G00, G02, G03.	Modal.	No.
<u>G02</u> [▶ 38]	Clockwise interpolation mode: Circular or helical. Applying current velocity. Reset by G00, G01, G03.	Modal.	No.
<u>G03</u> [▶ 38]	Counterclockwise interpolation mode: Circular or helical. Applying current velocity. Reset by G00, G01, G02.	Modal.	No.
<u>G04</u> [▶ 40]	Defines a dwell time, i.e. suspends machining for a given duration.	Nonmodal.	No.

Workingplane Selection

Com-mand	Description	Modal or Non-modal	Default
<u>G17</u> [▶ 44]	Selects XY-plane as workingplane.	Modal.	Default.
<u>G18</u> [▶ 44]	Selects ZX-plane as workingplane.	Modal.	No.
<u>G19</u> [▶ 44]	Selects YZ-plane as workingplane.	Modal.	No.

Delete Distance to go

Com-mand	Description	Modal or Non-modal	Default
<u>G31</u> [▶ 40]	Deletes Distance to go.	Nonmodal.	No.

Deactivation and Activation of Tool Radius Compensation

Com-mand	Description	Modal or Non-modal	Default
<u>G40</u> [▶ 42]	Deactivates Tool Radius Compensation (TRC). With/After a G40 command it is mandatory to program at least one geometry element.	Modal.	Default.
<u>G41</u> [▶ 42]	Activates Tool Radius Compensation (TRC). Left.	Modal.	No.
<u>G42</u> [▶ 42]	Activates Tool Radius Compensation (TRC). Right.	Modal.	No.

Set, Deactivate and Activate Zero-Offset-Shift Translations

Com-mand	Description	Modal or Non-modal	Default
<u>G53</u> [▶ 41]	Deactivates any zero-offset-shift translation.	Modal.	Default.
<u>G54..G57</u> [▶ 41]	Activates the zero-offset-shift associated with the given G-Code. Activates the translation G58 and G59.	Modal.	No.
<u>G58, G59</u> [▶ 41]	Sets the translation associated with the given G-Code.	Modal.	No.

Command	Description
<u>zeroOffsetShift</u> <u>Set(g:= USInt,</u> [▶ 82] <u>x:= LReal,</u> [▶ 82] <u>y:= LReal</u> [▶ 82], <u>z:= LReal)</u> [▶ 82]	Sets the translation for G-Code g where g must be one of the numbers 54, 55, 56 or 57.

Accurate Stop

Com-mand	Description	Modal or Non-modal	Default
<u>G09</u> [▶ 40]	Accurate stop.	Nonmodal.	No.
<u>G60</u> [▶ 40]	Accurate stop.	Modal.	No.

Set Unit for Length and Speed

Com-mand	Description	Modal or Non-modal	Default
<u>G70</u> [▶ 45]	Sets the unit for lengths to inch. Does not affect the unit for velocity.	Modal.	No.
<u>G71</u> [▶ 45]	Sets the unit for lengths to millimeter. Does not affect the unit for velocity.	Modal.	Default.
<u>G700</u> [▶ 45]	Sets the unit for lengths to inch. Also applies to the interpretation of velocity.	Modal.	No.

Command	Description	Modal or Non-modal	Default
G710 [▶ 45]	Sets the unit for lengths to millimeter. Also applies to the interpretation of velocity.	Modal.	Default.

Switch to Absolute or Relative Coordinates

Command	Description	Modal or Non-modal	Default
G90 [▶ 47]	Switches to absolute programming.	Modal.	Default.
G91 [▶ 47]	Switches to incremental programming.	Modal.	No.

IJK

Command	Description	Modal or Non-modal	Default
I<vx> J<vy> K<vz> [▶ 38]	Center point is <code>currentPoint + [vx,vy,vz]</code> . Current length unit is used for <code>vx,vy,vz</code> . I used by G4 defines a duration.	Modal.	Default: Center point is <code>currentPoint + [0,0,0]</code> . The IJK-parameters are optional.

M-Functions

Command	Description
M<v> [▶ 47]	Triggers the M-function <code>v</code> . The timing behavior depends on the definition of <code>v</code> in the System Manager of TwinCAT. There must not be more than one M-function of type handshake in a block.
M2	Predefined M-function. Signals program end.
M30	Predefined M-function. Signals program end.

Tool Orientation

Command	Description	Modal or Non-modal	Default
P<v> [▶ 44]	Switches tool orientation.	Modal.	No.

Set Block Number

Command	Description
N<v> [▶ 48]	Block number.

Set Radius

Com-mand	Description	Modal or Non-modal	Default
<u>U<v></u> [▶ 38]	Sets the radius within the context of G02 or G03 to v . Uses current length unit for v.	Modal.	No.

Set Cartesian Coordinate

Com-mand	Description	Modal or Non-modal	Default
<u>X<v></u> [▶ 48]	Sets the X-coordinate of the next point to v. Uses current length unit for v.		
<u>Y<v></u> [▶ 48]	Sets the Y-coordinate of the next point to v. Uses current length unit for v.		
<u>Z<v></u> [▶ 48]	Sets the Z-coordinate of the next point to v. Uses current length unit for v.		

Auxiliary Axes

Com-mand	Description	Modal or Non-modal	Default
<u>Q<i>=<v></u> > [▶ 48]	Sets label for auxiliary axis.	Modal.	No.

Set Orientation Angle

Com-mand	Description	Modal or Non-modal	Default
<u>A<v></u> [▶ 48]	Sets the A-angle of the next orientation to v. Uses current length unit for v.	Nonmodal, but may influence succeeding blocks.	No.
<u>B<v></u> [▶ 48]	Sets the B-angle of the next orientation to v. Uses current length unit for v.	Nonmodal, but may influence succeeding blocks.	No.
<u>C<v></u> [▶ 48]	Sets the C-angle of the next orientation to v. Uses current length unit for v.	Nonmodal, but may influence succeeding blocks.	No.

Trigonometric

Command	Description
<u>SIN(x)</u> [▶ 59]	Returns the sine of x; x in radians.
<u>COS(x)</u> [▶ 59]	Returns the cosine of x; x in radians.
<u>TAN(x)</u> [▶ 59]	Returns the tangent of x; x in radians.
<u>ASIN(x)</u> [▶ 59]	Returns the arc sine of x; x in radians.
<u>ACOS(x)</u> [▶ 59]	Returns the arc cosine of x; x in radians.
<u>ATAN(x)</u> [▶ 59]	Returns the arc tangent of x; x in radians.
<u>ATAN2(y, x)</u> [▶ 59]	Returns the arc tangent of y/x; y/x in radians.

Arithmetic

Command	Description
<u>ABS(x)</u> [▶ 59]	Returns the absolute value of x .
<u>SQRT(x)</u> [▶ 59]	Returns the square root of x .
<u>LN(x)</u> [▶ 59]	Returns the natural logarithm of x .
<u>LOG(x)</u> [▶ 59]	Returns the decimal logarithm of x .
<u>EXP(x)</u> [▶ 59]	Returns e raised to the power of x .
<u>ADD(x1, x2, ...)</u> [▶ 59]	Returns the sum of all parameters.
<u>MUL(x1, x2, ...)</u> [▶ 59]	Returns the product of all parameters.
<u>SUB(x, y)</u> [▶ 59]	Returns the difference $x - y$.
<u>DIV(x, y)</u> [▶ 59]	Returns the quotient x / y .
<u>MOD(x, y)</u> [▶ 59]	Returns the remainder of the integer division x / y .
<u>EXPT(x, y)</u> [▶ 59]	Returns x raised to the power of y .

Arithmetic Parameters

Command	Description
<u>rSet(index := LINT, value := LREAL)</u> [▶ 66]	Assigns an R -parameter value.
<u>rGet(index := LINT)</u> [▶ 66]	Extracts an R -parameter value.

Bit Shift and Bit Rotation

Command	Description
<u>SHL(x, y)</u> [▶ 62]	Returns the bitstring x shifted left by y bits.
<u>SHR(x, y)</u> [▶ 62]	Returns the bitstring x shifted right by y bits.
<u>ROL(x, y)</u> [▶ 62]	Returns the bitstring x rotated left by y bits.
<u>ROR(x, y)</u> [▶ 62]	Returns the bitstring x rotated right by y bits.

Logical Operations

Command	Description
<u>AND(x1, x2, ...)</u> [▶ 63]	Returns the bitwise Logical And of all parameters.
<u>OR(x1, y2, ...)</u> [▶ 63]	Returns the bitwise Logical Or of all parameters.
<u>XOR(x1, x2, ...)</u> [▶ 63]	Returns the bitwise Logical Exclusive Or of all parameters.
<u>NOT(x)</u> [▶ 63]	Returns the bitwise complement of x .

Selection (Conditional Expressions)

Command	Description
<u>SEL(cond, x1, x2)</u> [▶ 64]	Returns $x1$ if $cond$ is <code>FALSE</code> , and $x2$ otherwise.
<u>MUX(select, x0, x1, ..., xN)</u> [▶ 64]	Returns $x <select>$.

Min, Max and Limit

Command	Description
<code>MAX(x1, x2, ...)</code> [▶ 64]	Returns the maximum of all parameters.
<code>MIN(x1, x2, ...)</code> [▶ 64]	Returns the minimum of all parameters.
<code>LIMIT(min, in, max)</code> [▶ 64]	Returns <code>in</code> if it lies in the interval <code>[min, max]</code> . Otherwise, the violated bound (<code>min</code> or <code>max</code>) is returned.

Comparison

Command	Description
<code>GT(x, y)</code> [▶ 65]	Returns <code>TRUE</code> if <code>x</code> is larger than <code>y</code> .
<code>GE(x, y)</code> [▶ 65]	Returns <code>TRUE</code> if <code>x</code> is not smaller than <code>y</code> .
<code>EQ(x, y)</code> [▶ 65]	Returns <code>TRUE</code> if <code>x</code> and <code>y</code> are equal.
<code>LE(x, y)</code> [▶ 65]	Returns <code>TRUE</code> if <code>x</code> is not larger than <code>y</code> .
<code>LT(x, y)</code> [▶ 65]	Returns <code>TRUE</code> if <code>x</code> is smaller than <code>y</code> .
<code>NE(x, y)</code> [▶ 65]	Returns <code>TRUE</code> if <code>x</code> and <code>y</code> are not equal.

Strings and Messages

Command	Description
<code>toString(<arg0>, ..., <argN>): S</code> <code>TRING</code> [▶ 67]	Converts and concatenates the given arguments to one string.
<code>msg(str:= String[81])</code> [▶ 67]	Sends the given message to the message list of TwinCAT.

Transformations

Command	Description
<code>transRotX(angle:= LReal)</code> [▶ 68] <code>transRotY(angle:= LReal)</code> [▶ 68] <code>transRotZ(angle:= LReal)</code> [▶ 68]	Rotation around the respective axis by the given angle in the user-defined angle unit.
<code>transRotA(x:= LReal</code> [▶ 68], <code>y:= LReal</code> , [▶ 68] <code>z:= LReal</code> [▶ 68], <code>angle:= LReal)</code> [▶ 68]	Rotate around vector <code>[x, y, z]</code> by the given angle.
<code>transMirrorX()</code> [▶ 68] <code>transMirrorY()</code> [▶ 68] <code>transMirrorZ()</code> [▶ 68]	Mirror with respect to the X-direction, Y-direction or Z-direction relative to the origin of the current PCS.
<code>transScale(factor:= LReal)</code> [▶ 68]	Scales the coordinate system by the <code>factor</code> in the X-dimension, Y-dimension and Z-dimension.
<code>transScaleAxis(axisNo :=</code> <code>axisIndex, factor := value)</code> [▶ 68]	Scales the selected path axis (<code>axisNo</code>) by the factor.
<code>transTranslate(x:= LReal</code> , [▶ 68] <code>y:= LReal</code> , [▶ 68] <code>z:= LReal)</code> [▶ 68]	Translate by vector <code>[x, y, z]</code> .
<code>transPop()</code> [▶ 68]	Pops a transformation from the stack of transformations.
<code>transDepth(): UInt</code> [▶ 68]	Yields the depth of the stack of transformations, i.e. the number of active transformations.
<code>transRestore(depth:= UInt)</code> [▶ 68]	Reduces the stack of transformations to the given depth.

Movement

Command	Description
<code>moveCircle3d(cx:= LReal, [▶ 75] cy:= LReal, [▶ 75] cz:= LReal, [▶ 75] nx:= LReal [▶ 75], ny:= LReal [▶ 75], nz:= LReal [▶ 75], angle:= LReal [▶ 75], height:= LReal) [▶ 75]</code>	Move circular by rotating around the center cx, cy, cz and the normal vector nx, ny, nz by the given <code>angle</code> . If <code>height</code> is nonzero, a helix is described. The rotation is performed according to the right hand rule.

Centerpoint Correction

Command	Description
<code>centerpointCorrectionSet(on:= bool) [▶ 75]</code>	Activates the centerpoint correction for circles. Used for circles that are defined by centerpoint programming.
<code>centerpointCorrectionLimitSet(li mit:= LReal) [▶ 75]</code>	Sets the precision limit for the centerpoint of circles.

Tools

Command	Description
<code>toolParamSet(tidx:= USInt, [▶ 76] col:= USInt [▶ 76], val:= LReal) [▶ 76]</code>	Set a parameter of the tool <code>tidx</code> (1..255) to <code>val</code> . The parameter is identified by <code>col</code> (0..15).
<code>toolParam(tidx:= USInt [▶ 76], col:= USInt): LReal [▶ 76]</code>	Yields the given tool parameter.
<code>toolSet(index:= USInt [▶ 76], nr:= Int [▶ 76], tooltype:= ToolType, [▶ 76] length:= LReal [▶ 76], radius:= LReal, [▶ 76] lengthAdd:= LReal, [▶ 76] radiusAdd:= LReal [▶ 76], offsetX:= LReal [▶ 76], offsetY:= LReal [▶ 76], offsetZ:= LReal) [▶ 76]</code>	Set all parameters of a tool.
<code>ToolType [▶ 76]</code>	Enumeration of tool types.

Tool Radius Compensation

Command	Description
<code>trcApproachDepartSet(approac hRadius:= LReal [▶ 80], approachAngle:= LReal [▶ 80], departRadius:= LReal, [▶ 80] departAngle:= LReal) [▶ 80]</code>	Configures the approach and depart behavior to use an arc of given radius and angle.
<code>trcOffsetSet(offset:= LReal) [▶ 80]</code>	Configures the amount of segment extension that is used to close gaps.
<code>trcLimitSet(offset:= LReal) [▶ 80]</code>	Configures the lookahead that is used for collision elimination.
<code>trcParam(): TrcParamType [▶ 80]</code>	Returns the current configuration as a structure value.

Command	Description
<code>trcParamSet(param:= TrcParamType)</code> [► 80]	Configures the tool radius compensation. Summarizes <code>trcApproachDepartSet</code> , <code>trcOffsetSet</code> and <code>trcLimitSet</code> .
<code>TrcParamType</code> [► 80]	Structure containing all configuration parameters of the tool radius compensation.
<code>collisionElimination(nx:= LReal [► 80], ny:= LReal, [► 80] nz:= LReal [► 80], limit:= ULLnt) [► 80]</code>	Activates collision elimination with respect to the plane of the normal vector <code>nx</code> , <code>ny</code> , <code>nz</code> .
<code>collisionEliminationFlush()</code> [► 80]	To ignore conflicts between the path preceding the call and the path succeeding the call.

Command	Description	Modal or Non-modal	Default
<code>G40</code> [► 42]	Deactivates Tool Radius Compensation (TRC).	Modal.	Default.
<code>G41</code> [► 42]	Activates Tool Radius Compensation (TRC). Left.	Modal.	No.
<code>G42</code> [► 42]	Activates Tool Radius Compensation (TRC). Right.	Modal.	No.

Synchronization

Command	Description
<code>sync()</code> [► 78]	Synchronizes the interpreter with the associated NC-channel.
<code>wait()</code> [► 78]	Waits for a <code>GoAhead</code> -signal from the PLC.

Query of Axes

Command	Description
<code>queryAxes()</code> [► 78]	Set the MCS coordinates of the interpreter to the actual coordinates of the physical axes.

Current Point

Command	Description
<code>frameGet(x:= LReal [► 79], y:= LReal [► 79], z:= LReal [► 79], a:= LReal [► 79], b:= LReal [► 79], c:= LReal) [► 79]</code>	Store the current frame of the PCS in <code>x</code> , <code>y</code> , <code>z</code> and <code>a</code> , <code>b</code> , <code>c</code> .
<code>qAxisGet(q1:= LReal, [► 79] q2:= LReal [► 79], q3:= LReal, [► 79] q4:= LReal, [► 79] q5:= LReal) [► 79]</code>	Store the current values of Q-axes <code>q1</code> to <code>q5</code> .

Suppression of G-Code Blocks

Command	Description
<code>disableMask():= LWord [► 81]</code>	Yields the current value of the disable mask.

Command	Description
<code>disableMaskSet(mask:= LWord)</code> [▶ 81]	Sets the internal disable mask to the given value.

Units

Command	Description
<code>unitAngleSet(unitAngle:= UnitAngle)</code> [▶ 83]	Set the unit for angles to <code>unitAngle</code> .
<code>UnitAngle</code> [▶ 83]	Enumeration of unit angles.
<code>unitLengthSet(unitLength:= UnitLength)</code> [▶ 83]	Set the unit for lengths to <code>unitLength</code> .
<code>UnitLength</code> [▶ 83]	Enumeration of unit lengths.
<code>unitTimeSet(unitTime:= UnitTime)</code> [▶ 83]	Set the unit for time to <code>unitTime</code> .
<code>UnitTime</code> [▶ 83]	Enumeration of unit times.
<code>unitVelocitySet(unitLength:= UnitLength, unitTime:= UnitTime)</code> [▶ 83]	Set the unit for velocity to <code>unitLength/unitTime</code> .

Trigonometric (Unit Aware)

Command	Description
<code>gSin(angle:= LReal)</code> [▶ 84]	Returns the sine of the given <code>angle</code> where the current angle unit is used to interpret the angle.
<code>gCos(angle:= LReal)</code> [▶ 84]	Returns the cosine of the given <code>angle</code> where the current angle unit is used to interpret the angle.
<code>gTan(angle:= LReal)</code> [▶ 84]	Returns the tangent of the given <code>angle</code> where the current angle unit is used to interpret the angle.
<code>gASin(val:= LReal)</code> [▶ 84]	Returns the arc sine of <code>val</code> in the current angle unit.
<code>gACos(val:= LReal)</code> [▶ 84]	Returns the arc cosine of <code>val</code> in the current angle unit.
<code>gATan(val:= LReal)</code> [▶ 84]	Returns the arc tangent of <code>val</code> in the current angle unit.
<code>gATan2(y:= LReal, x:= LReal)</code> [▶ 84]	Returns the arc tangent of <code>y/x</code> in the current angle unit.

Feed Mode

Command	Description
<code>feedModeSet(feedMode:= FeedModeType)</code> [▶ 85]	Set the feed mode.
<code>FeedModeType</code> [▶ 85]	Enumeration of feed mode types.

Feed Mode

Command	Description
<code>feedInterpolationSet(feedInterpolation:= FeedInterpolationType)</code> [▶ 85]	Set feed interpolation.
<code>FeedInterpolationType</code> [▶ 85]	Enumeration of feed interpolation types.

Streaming of Large G-Code Files

Command	Description
runFile(path:= string) [▶ 86]	Executes the plain G-Code that is contained in the G-Code file given by path.

Vertex Smoothing

Command	Description
smoothingSet [▶ 86] (mainType:= SmoothingMainType [▶ 86] , subType:= SmoothingSubType [▶ 86] , value:= LReal) [▶ 86]	Set the vertex smoothing behavior.
SmoothingMainType [▶ 86]	Enumeration of smoothing main types.
SmoothingSubType [▶ 86]	Enumeration of smoothing sub types.
autoAccurateStopSet [▶ 88]	Automatic accurate stop.

Dynamic Override

Command	Description
dynOverrideSet(value:= LReal) [▶ 92]	Set the dynamic override of axes to the given value.

Center Point Reference of Circles

Command	Description
circleCenterReferenceSet(value:= ReferenceType) [▶ 93]	Sets the center reference type for circles that are programmed by G02/G03.
ReferenceType [▶ 93]	Enumeration of reference types.

Dynamics Set

Command	Description
axisDynamicsSet [▶ 93]	Change in axis dynamics.
pathDynamicsSet [▶ 94]	Change in path dynamics.

Spline Interpolation

Command	Description
transBSpline [▶ 89]	Spline Interpolation.

4.10 Comparative Command Overview

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
ANG [▶ 144]		Nonmodal.	Contour line programming (angle).

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
CalcInvRot [▶ 145]		Nonmodal.	Calculates the inverse rotation of a vector.
CalcRot [▶ 145]		Nonmodal.	Calculates the rotation of a vector.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
CDOF [▶ 189]	collisionElimination [▶ 80] Supplying a zero vector.	Modal.	Bottleneck detection off.
CDON [▶ 189]	collisionElimination [▶ 80] Supplying a nonzero vector.	Modal.	Bottleneck detection on.
	collisionEliminationFlush [▶ 80]		This function can be called during active collision elimination to ignore any conflicts between the path preceding the call and the path succeeding the call.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
CFC [▶ 188]	feedModeSet(feedMode:= FeedModeType) [▶ 85]	Modal.	Constant velocity at the contour.
CFIN [▶ 188]	feedModeSet(feedMode:= FeedModeType) [▶ 85]	Modal.	Constant velocity in the interior circle.
CFTCP [▶ 188]	feedModeSet(feedMode:= FeedModeType) [▶ 85]	Modal.	Constant velocity of the tool center point.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
CIP [▶ 135]	moveCircle3D [▶ 75]	Nonmodal.	Circular interpolation. Move circular by rotating around the center cx, cy, cz and the normal vector nx, ny, nz by the given angle.
CPCOF [▶ 135]	centerpointCorrectionSet [▶ 75]	Modal.	Center point correction off.
CPCON [▶ 135]	centerpointCorrectionSet [▶ 75]	Modal.	Center point correction on.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
DelDTG [▶ 156]	G31 [▶ 40]	Nonmodal.	Delete distance to go.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
DYNQVR [▶ 172]	dynOverrideSet [▶ 92]	Modal.	Dynamic override.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
FCONST [▶ 138]	feedInterpolationSet(feedInterpolation:= fiConstant) [▶ 86]	Modal.	Constant feed programming.
FLIN [▶ 138]	feedInterpolationSet(feedInterpolation:= fiLinear) [▶ 86]	Modal.	Linear feed programming.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G00 [▶ 133]	G00 [▶ 37]	Modal.	Rapid traverse.
G01 [▶ 134]	G01 [▶ 38]	Modal.	Straight line interpolation.
G02 [▶ 135]	G02 [▶ 38]	Modal.	Clockwise circular interpolation.
G03 [▶ 135]	G03 [▶ 38]	Modal.	Counterclockwise circular interpolation.
G04 [▶ 137]	G04 [▶ 40]	Nonmodal.	Dwell time.
G09 [▶ 138]	G09 [▶ 40]	Nonmodal.	Accurate stop.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G17 [▶ 126]	G17 [▶ 44]	Modal.	Plane selection XY.
G18 [▶ 126]	G18 [▶ 44]	Modal.	Plane selection ZX.
G19 [▶ 126]	G19 [▶ 44]	Modal.	Plane selection YZ.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G40 [▶ 183]	G40 [▶ 42]	Modal.	No miller/ cutter radius compensation.
G41 [▶ 183]	G41 [▶ 42]	Modal.	Miller/ cutter radius compensation left.
G42 [▶ 183]	G42 [▶ 42]	Modal.	Miller/ cutter radius compensation right.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G53 [▶ 139]	G53 [▶ 41]	Modal.	Zero shift suppression.
G54 [▶ 139]	G54 [▶ 41]	Modal.	1 st adjustable zero shift.
G55 [▶ 139]	G55 [▶ 41]	Modal.	2 nd adjustable zero shift.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G56 [▶ 139]	G56 [▶ 41]	Modal.	3 rd adjustable zero shift.
G57 [▶ 139]	G57 [▶ 41]	Modal.	4 th adjustable zero shift.
G58 [▶ 139]	G58 [▶ 41]	Modal.	1 st programmable zero shift.
G59 [▶ 139]	G59 [▶ 41]	Modal.	2 nd programmable zero shift.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G60 [▶ 138]	G60 [▶ 40]	Modal.	Accurate stop.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G70 [▶ 128]	G70 [▶ 45]	Modal.	Dimensions in inch.
G71 [▶ 128]	G71 [▶ 45]	Modal.	Dimensions metric.
G74 [▶ 133]		Nonmodal.	Programmed traverse to reference point.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G90 [▶ 126]	G90 [▶ 47]	Modal.	Reference dimension notation.
G91 [▶ 126]	G91 [▶ 47]	Modal.	Incremental dimension notation.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
G700 [▶ 128]	G700 [▶ 45]	Modal.	Dimensions in inches with calculation of the feed.
G710 [▶ 128]	G710 [▶ 45]	Modal.	Dimensions metric with calculation of the feed.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
MOD [▶ 156]		Nonmodal.	Modulo movement.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
MSG [▶ 177]	msg [▶ 67]	Nonmodal.	Message from the NC program.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
NORM [▶ 188]		Nonmodal.	Orthogonal approach off the contour and orthogonal departure from the contour.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
P+ [▶ 126]	P<1> [▶ 48]	Modal.	Feed direction positive.
P- [▶ 126]	P<-1> [▶ 48]	Modal.	Feed direction negative.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
paramAutoAccurateStop	autoAccurateStopSet [▶ 88]	Modal.	Automatic accurate stop.
paramAxisDynamics [▶ 172]	axisDynamicsSet [▶ 93]	Modal.	Parameterization of the axis dynamics.
paramC1ReductionFactor [▶ 173]		Modal.	C1 reduction factor.
paramC2ReductionFactor [▶ 173]		Modal.	C2 reduction factor.
paramCircularSmoothing [▶ 154]		Modal.	Circular smoothing.
paramDevAngle [▶ 173]		Modal.	C0 reduction - deflection angle.
paramGroupVertex [▶ 154]		Modal.	Circular smoothing (old).
paramGroupDynamic [▶ 172]		Modal.	Path dynamics (old).
paramPathDynamics [▶ 172]	pathDynamicsSet [▶ 94]	Modal.	Path dynamics.
paramRadiusPrec [▶ 136]		Modal.	Circular accuracy.
paramSplineSmoothing [▶ 152]	smoothingSet [▶ 86]	Modal.	Vertex smoothing. NC: Smoothing with Bezier Splines.
paramVertexSmoothing [▶ 149]	smoothingSet [▶ 86]	Modal.	Smoothing of segment transitions.
	transBSpline [▶ 89]	Modal.	Spline interpolation.
paramVelocityJump [▶ 173]		Modal.	C0 reduction - maximum step change in velocity.
paramVelocityMin [▶ 175]		Modal.	Minimum velocity.
paramZeroShift [▶ 139]	zeroOffsetShiftSet [▶ 139]	Modal.	Parameterization of the configurable zero shift.
PathAxesPos [▶ 176]	frameGet , [▶ 79] qAxisGet [▶ 79]	Nonmodal.	Reads the actual position.
Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
AROT [▶ 145]	transRotA [▶ 68]	Modal.	Rotation additive.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
ROT [▶ 145]	transRotX , transRotY , transRotZ [▶ 68]	Modal.	Absolute rotation.
RotExOff [▶ 145]		Modal.	Extended rotation function off.
RotExOn [▶ 145]		Modal.	Extended rotation function on.
RotVec [▶ 145]		Nonmodal.	Calculation routine for rotating a vector.
RToDwordGetBit [▶ 130]		Modal.	Converts an R-parameter to DWord and checks whether a defined bit is set.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
rParam [▶ 130]	rSet(index := LINT, value := LREAL [▶ 66])	Nonmodal.	Assigning a Value to an R-Parameter.
rParam [▶ 130]	rGet(index := LINT) [▶ 66]	Nonmodal.	Reading an R-Parameter Value.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
SEG [▶ 144]		Nonmodal.	Contour line programming (segment length).
skip VirtualMovements [▶ 177]		Modal.	Skip virtual movements.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
ToolOffsetIncOff [▶ 180]		Modal.	Cartesian tool displacement and length compensation is not applied under G91.
ToolOffsetIncOn [▶ 180]		Modal.	Cartesian tool displacement and length compensation is applied under G91.
ToolParam [▶ 177]	toolParamSet [▶ 76]	Modal.	Set a tool parameter. NC: Writing and reading of tool parameters.
ToolParam [▶ 177]	toolParam [▶ 76]	Modal.	Yields the given tool parameter. NC: Writing and reading of tool parameters.
ToolParam [▶ 177]	toolSet [▶ 76]	Modal.	Set all parameters of a tool. NC: Writing and reading of tool parameters.
TPM [▶ 142]		Nonmodal.	Target position monitoring.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
ZeroShiftIncOff [▶ 139]		Modal.	Zero shift is not applied under G91.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
ZeroShiftIncOn [▶ 139]		Modal.	Zero shift is applied under G91.

Command Classic Interpreter	Command GST Interpreter	Modal or Non-modal	Description
Mirror [▶ 148]	transMirrorX , transMirrorY , transMirrorZ [▶ 68]	Modal.	Mirroring the coordinate system.
	transScale [▶ 68]	Modal.	Scales the coordinate system by the <code>factor</code> in the X-dimension, Y-dimension and Z-dimension.
	transScaleAxis [▶ 68]	Modal.	Scales the selected path axis (<code>axisNo</code>) by the factor.
	transDepth [▶ 68]	Nonmodal.	Yields the depth of the stack of transformations.
	transRestore [▶ 68]	Modal.	Reduces the stack of transformations to the given depth.
	transPop [▶ 68]	Modal.	Pops a transformation from the stack of transformations.

Command Classic Interpreter	Command GST Interpreter	Description
CIP [▶ 135]	moveCircle3d [▶ 75]	Move circular by rotating around the center <code>cx,cy,cz</code> and the normal vector <code>nx,ny,nz</code> by the given <code>angle</code> .

Command Classic Interpreter	Command GST Interpreter	Description
	queryAxes [▶ 78]	Set the MCS (machine coordinate system) coordinates of the interpreter to the actual coordinates of the physical axes.

Command Classic Interpreter	Command GST Interpreter	Description
	trcApproachDepartSet [▶ 80]	Configures the approach and depart behavior to use an arc of given radius and angle.
	trcOffsetSet [▶ 80]	Configures the amount of segment extension that is used to close gaps.
	trcLimitSet [▶ 80]	Configures the lookahead that is used for collision elimination.
	trcParam [▶ 80]	Returns the current configuration as a structure value.
	trcParamSet [▶ 80]	Configures the tool radius compensation.
	TrcParamType [▶ 80]	This structure contains all configuration parameters of the tool radius compensation.

Command Classic Interpreter	Command GST Interpreter	Description
	disableMask [▶ 81]	Yields the current value of the disable mask.
Block Skipping [▶ 123] /	disableMaskSet [▶ 81]	Sets the internal disable mask to the given value.

Command Classic Interpreter	Command GST Interpreter	Description
L [▶ 170]	Userdefined Functions [▶ 57]	Call of a subroutine.
	#include [▶ 31]	Directive inserts the contents of another file. Typically, it is used to "import" commonly used code like e.g. libraries.
	runFile [▶ 86]	Streaming of large G-Code files.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@40 [▶ 130]	@40 Kn Rn Rm ...		Save register on the stack.
@41 [▶ 130]	@41 Rn Rm		Save register on the stack.
@42 [▶ 130]	@42 Kn ... Rm Rn		Restore register from stack.
@43 [▶ 130]	@43 Rm Rn		Restore register from stack.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@100 [▶ 167]	@100 K±n @100 Rm		Unconditional jump.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@111 [▶ 167]	@111 Rn K/Rn Km ...	CASE OF	Case block.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@121 [▶ 167]	@121 Rn K/Rn Kn	IF-THEN-ELSIF-ELSE;CASE OF [▶ 56]	Jump if unequal.
@122 [▶ 167]	@122 Rn K/Rn Kn	IF-THEN-ELSIF-ELSE;CASE OF [▶ 56]	Jump if equal.
@123 [▶ 167]	@123 Rn K/Rn Kn	IF-THEN-ELSIF-ELSE;CASE OF [▶ 56]	Jump if less or equal.
@124 [▶ 167]	@124 Rn K/Rn Kn	IF-THEN-ELSIF-ELSE;CASE OF [▶ 56]	Jump if less.
@125 [▶ 167]	@125 Rn K/Rn Kn	IF-THEN-ELSIF-ELSE;CASE OF [▶ 56]	Jump if greater or equal.
@126 [▶ 167]	@126 Rn K/Rn Kn	IF-THEN-ELSIF-ELSE;CASE OF [▶ 56]	Jump if greater.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@131 [▶ 169]	@131 Rn K/Rn Kn	WHILE [▶ 56]	Loop while equal.
@132 [▶ 169]	@132 Rn K/Rn Kn	WHILE [▶ 56]	Loop while unequal.
@133 [▶ 169]	@133 Rn K/Rn Kn	WHILE [▶ 56]	Loop while greater.
@134 [▶ 169]	@134 Rn K/Rn Kn	WHILE [▶ 56]	Loop while greater or equal.
@135 [▶ 169]	@135 Rn K/Rn Kn	WHILE [▶ 56]	Loop while less.
@136 [▶ 169]	@136 Rn K/Rn Kn	WHILE [▶ 56]	Loop while less or equal.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@141 [▶ 169]	@141 Rn K/Rn Kn	REPEAT [▶ 56]	Repeat until equal.
@142 [▶ 169]	@142 Rn K/Rn Kn	REPEAT [▶ 56]	Repeat until unequal.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@143 > 169	@143 Rn K/Rn Kn	REPEAT > 56	Repeat until greater.
@144 > 169	@144 Rn K/Rn Kn	REPEAT > 56	Repeat until greater or equal.
@145 > 169	@145 Rn K/Rn Kn	REPEAT > 56	Repeat until less.
@146 > 169	@146 Rn K/Rn Kn	REPEAT > 56	Repeat until less or equal.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@151 > 169	@151 Rn K/Rn Kn	FOR > 56	FOR_TO loop.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@161 > 169	@161 Rn K/Rn Kn	FOR > 56	FOR_DOWNTO loop.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@200	@200 Rn		Delete a variable.
@202	@202 Rn Rm		Swap two variables.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@302	@302 K/R/Pn K/R/Pn R/Pn		Read machine data bit.
@361 > 176	@361 Rn Km		Read machine-related actual axis value.
@372	@372 Rn		Extract the NC-Channel-ID and store it in a variable.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@402 > 135	@402 K/R/Pn K/R/Pn K/R/Pn	circleCenterReferenceSet > 93	Write machine data bit.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@610	@610 Rn Rn	ABS > 59	Find the absolute value of a variable.
@613	@613 Rn Rn	SQRT > 59	Find the square root of a variable.
@614	@614 Rn Rm Rm	SQRT (a^2 + b^2)	Find the square root of the sum of the squares of two variables ! x := sqrt(a^2 + b^2);.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@620 > 169	@620 Rn	!var := var+1;	Increment variable.
@621	@621 Rn	!var := var-1;	Decrement variable.
@622	@622 Rn		Find integer part of a variable.
Command Classic Interpreter	Versions	Command GST Interpreter	Description
@630 > 130	@630 Rn Rm	SIN > 59	Find the sine of a variable.
@630 > 130	@630 Rn Rm	gSin > 84	Find the sine of a variable.

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@631 [▶ 130]	@631 Rn Rm	COS [▶ 59]	Find the cosine of a variable.
@631 [▶ 130]	@631 Rn Rm	gCos [▶ 84]	Find the cosine of a variable.
@632 [▶ 130]	@632 Rn Rm	TAN [▶ 59]	Find the tangent of a variable.
@632 [▶ 130]	@632 Rn Rm	gTan [▶ 84]	Find the tangent of a variable.
@633 [▶ 130]	@633 Rn Rm		Find the cotangent of a variable.
@634 [▶ 130]	@634 Rn Rm	ASIN [▶ 59]	Find the arc sine of a variable.
@634 [▶ 130]	@634 Rn Rm	gASin [▶ 84]	Find the arc sine of a variable.
@635 [▶ 130]	@635 Rn Rm	ACOS [▶ 59]	Find the arc cosine of a variable.
@635 [▶ 130]	@635 Rn Rm	gACos [▶ 84]	Find the arc cosine of a variable.
@636 [▶ 130]	@636 Rn Rm	gATan [▶ 84]	Find the arc tangent of a variable.
		gATan2 [▶ 84]	Returns the arc tangent of y/x .

Command Classic Interpreter	Versions	Command GST Interpreter	Description
@714 [▶ 166]	@714	sync() [▶ 78]	Decoder stop.
@716 [▶ 166]	@716	A combination of sync() [▶ 78] and queryAxes() [▶ 78] replaces the former @716-command.	Decoder stop with rescan of the axis positions.
@717 [▶ 166]	@717	wait() [▶ 78]	Decoder stop with external trigger event.

Command Classic Interpreter	Command GST Interpreter	Description
	LN(x) [▶ 59]	Returns the natural logarithm of x .
	LOG(x) [▶ 59]	Returns the decimal logarithm of x .
	EXP(x) [▶ 59]	Returns e raised to the power of x .
	ADD(x1, x2, ...) [▶ 59]	Returns the sum of all parameters.
	MUL(x1, x2, ...) [▶ 59]	Returns the product of all parameters.
	SUB(x, y) [▶ 59]	Returns the difference $x-y$.
	DIV(x, y) [▶ 59]	Returns the quotient x/y .
	MOD(x, y) [▶ 59]	Returns the remainder of the integer division x/y .
	EXPT(x, y) [▶ 59]	Returns x raised to the power of y .

Command Classic Interpreter	Command GST Interpreter	Description
	GT(x, y) [▶ 65]	Returns TRUE if x is larger than y .
	GE(x, y) [▶ 65]	Returns TRUE if x is not smaller than y .
	EQ(x, y) [▶ 65]	Returns TRUE if x and y are equal.
	LE(x, y) [▶ 65]	Returns TRUE if x is not larger than y .
	LT(x, y) [▶ 65]	Returns TRUE if x is smaller than y .
	NE(x, y) [▶ 65]	Returns TRUE if x and y are not equal.

Command Class- sic Interpreter	Command GST In- terpreter	Description
	<code><nativeType> to <nativeType>(x)</code> [▶ 59] <code>to <nativeType>(x)</code> [▶ 59]	Explicit conversion between the given native types.

Logical Operations

Command Class- sic Interpreter	Command GST In- terpreter	Description
	<code>AND(x1, x2, ...)</code> [▶ 63]	Returns the bitwise Logical And of all parameters.
	<code>OR(x1, y2, ...)</code> [▶ 63]	Returns the bitwise Logical Or of all parameters.
	<code>XOR(x1, x2, ...)</code> [▶ 63]	Returns the bitwise Logical Exclusive Or of all parameters.
	<code>NOT(x)</code> [▶ 63]	Returns the bitwise complement of x .

Min, Max and Limit

Command Class- sic Interpreter	Command GST In- terpreter	Description
	<code>MAX(x1, x2, ...)</code> [▶ 64]	Returns the maximum of all parameters.
	<code>MIN(x1, x2, ...)</code> [▶ 64]	Returns the minimum of all parameters.
	<code>LIMIT(min, in, max)</code> [▶ 64]	Returns in if it lies in the interval $[min, max]$. Otherwise, the violated bound (min or max) is returned.

Rotation

Command Class- sic Interpreter	Command GST In- terpreter	Description
	<code>ROL(x, y)</code> [▶ 62]	Returns the bitstring x rotated left by y bits.
	<code>ROR(x, y)</code> [▶ 62]	Returns the bitstring x rotated right by y bits.

Selection (Conditional Expressions)

Command Class- sic Interpreter	Command GST In- terpreter	Description
	<code>SEL(cond, x1, x2)</code> [▶ 64]	Returns $x1$ if $cond$ is <code>FALSE</code> , and $x2$ otherwise.
	<code>MUX(select, x0, x1, ..., xN)</code> [▶ 64]	Returns $x<select>$.

Shift

Command Class- sic Interpreter	Command GST In- terpreter	Description
	<code>SHL(x, y)</code> [▶ 62]	Returns the bitstring x shifted left by y bits.
	<code>SHR(x, y)</code> [▶ 62]	Returns the bitstring x shifted right by y bits.

Units

Command Classic Interpreter	Command GST Interpreter	Description
	unitAngleSet(unitAngle:= UnitAngle) [▶ 83]	Set the unit for angles to <code>unitAngle</code> .
	UnitAngle [▶ 83]	Enumeration of unit angles.
	unitLengthSet(unitLength:= UnitLength) [▶ 83]	Set the unit for lengths to <code>unitLength</code> .
	UnitLength [▶ 83]	Enumeration of unit lengths.
	unitTimeSet(unitTime:= UnitTime) [▶ 83]	Set the unit for time to <code>unitTime</code> .
	UnitTime [▶ 83]	Enumeration of unit times.
	unitVelocitySet(unitLength:= UnitLength, unitTime:= UnitTime) [▶ 83]	Set the unit for velocity to <code>unitLength/unitTime</code> .

5 Classic Dialect Reference Manual

5.1 Basic Principles of NC Programming

5.1.1 Structure of an NC Program

An NC program is a text that is normally stored as a sequence of ASCII codes in a file on the hard disk. It consists of a sequence of NC blocks separated by line breaks (Return). Usually it is executed by being interpreted and worked through, character by character and line by line.

Program structure

The NC program is thus composed of three parts

- Program start (optional)
- Number of blocks
- Program end

Program start

At the beginning of an NC program the character "%" can represent the start of the program. The name of the program is then found following this character. The block for the program start does not necessarily have to be programmed.

Sample:

```
% Test1 (program start)
N10 G0 X100 Y100 Z0
M30 (program end)
```

NC block

Each NC block consists of one or several NC words, or even of none (an empty line), separated by spaces or tab characters. It is therefore not possible to use a space within a word.

Sample:

```
N10 G0 X100 Y100 Z0
```

NC word

The first character of an NC word specifies its meaning. It is either a letter or a special character.

Upper/lower case has, in general, no significance. Uniform use of upper case is, however, recommended for the sake of better readability. The optional following characters specify the meaning more precisely, or supply parameters for the execution.

In order to manage with such a limited supply of characters, an expression is not available for every variation of every function. It is rather the case that the significance and effect of many NC words is determined partly by the context. This can be a matter of the foregoing words in the block, but it can also depend on previous NC blocks. In a few cases the effect of NC words even depends on the machine data.

Program end

The end of the program is indicated by an M-function. Either M2 or M30 is used for this.

Effective Duration of Words

Commands such as `G0 [▶ 133]` or `G17 [▶ 126]`, that have effects beyond the end of the block, are known, according to DIN 66025, as **modal**. These commands are effective as long as they are neither cancelled nor altered by another command.

Comments

If either parts of an NC block, or the whole of it, is not to be interpreted, the region concerned is to be placed within curved brackets.

Sample:

```
N10 G0 X100 (comment)
```

Note A comment ends with the closing bracket, or, at the latest, at the end of the block. This means that a comment can't continue over a number of lines. Nested comments are also not possible.

Block numbers

Each block can be identified by a block number. The block number is accompanied by "N" for subordinate blocks and ":" for main blocks.

Note The block number is not essential. A block not identified by a block number can not, however, be used as the target of a jump command. An error message, moreover, can only approximately report the location of the error (the last block number).

5.1.2 Block Skipping

It is often useful if not all blocks of a program are always executed. This makes it possible to implement similar processes with a single program.

In such cases, the blocks that belong to one variant are given a block skipping identifier. This must be written at the start of the block, and consists of a slash "/".

If several variants are required, the slash is extended with line information (0..15), for instance "/12". The line information (where "/" is equivalent to "/0") selects a bit from a word in the channel interface from the PLC to the NC. If this bit is **set**, the block is not interpreted.

In the NC the variable '*mSkipLine*' is evaluated for this purpose, which can be found among the inputs in the cyclic channel interface. The counterpart in the PLC can be found in the outputs under '*nSkipLine*' [▶ 323] (*previously: nSkipBlock*) (see TwinCAT PLC library: NCI Interpreter).

If one of a number of variants is to be active, all the other suppressions must be set. Then only those blocks remain active that have either no identifier, or that have the desired identifier.

● Active time of block skipping

i The interpreter works an indeterminate number of blocks in advance of the execution. Block skipping can only operate correctly if it is set early enough (perhaps before the program starts), or if the interpreter is synchronized with the execution at a suitable location in the program (decoder stop [▶ 166]).

5.1.3 Look-Ahead

The actual velocity at the segment transition depends on a range of parameters. These include residual path length, dynamic parameters for the current segment, and (indirectly) the geometric angle at the segment transition.

Dynamic look-ahead (referred to as look-ahead below) ensures that the velocity can remain as high as possible at segment transitions. In the standard configuration 128 geometry entries are considered.

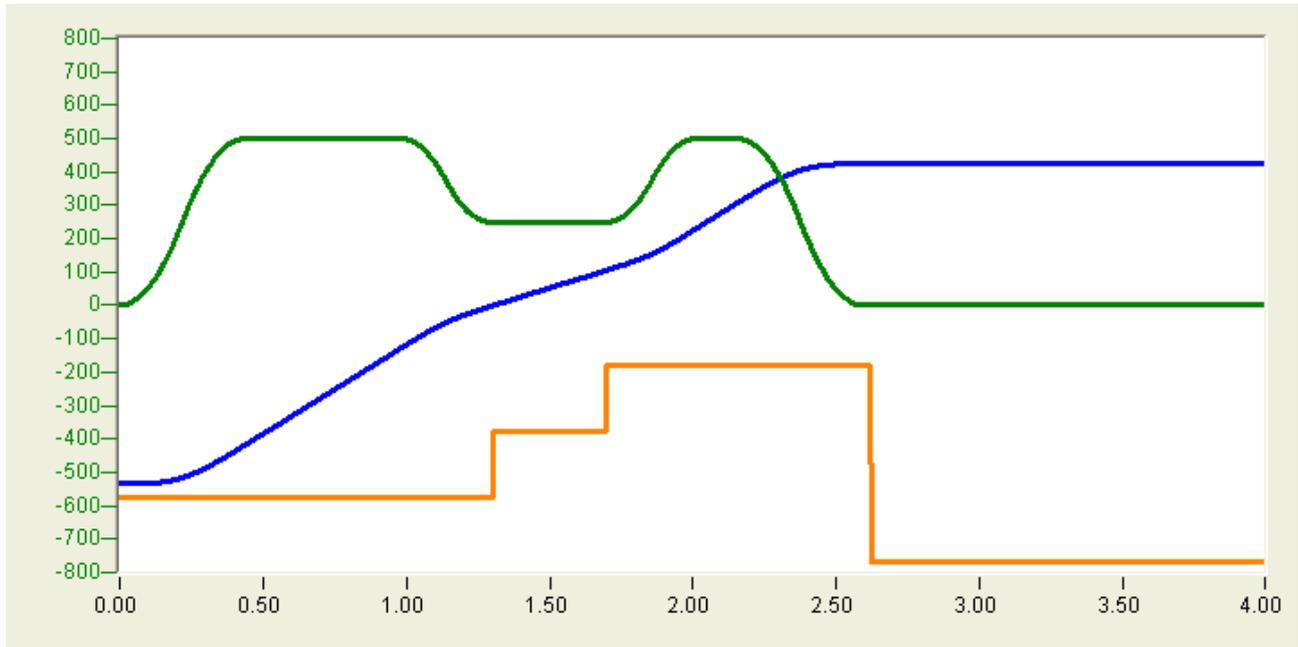
Without look-ahead the velocity is reduced to 0 at each segment transition (G60).

The number of geometry entries taken into account can be set in the [DXD parameters](#) [► 27].

Segments with different target velocity

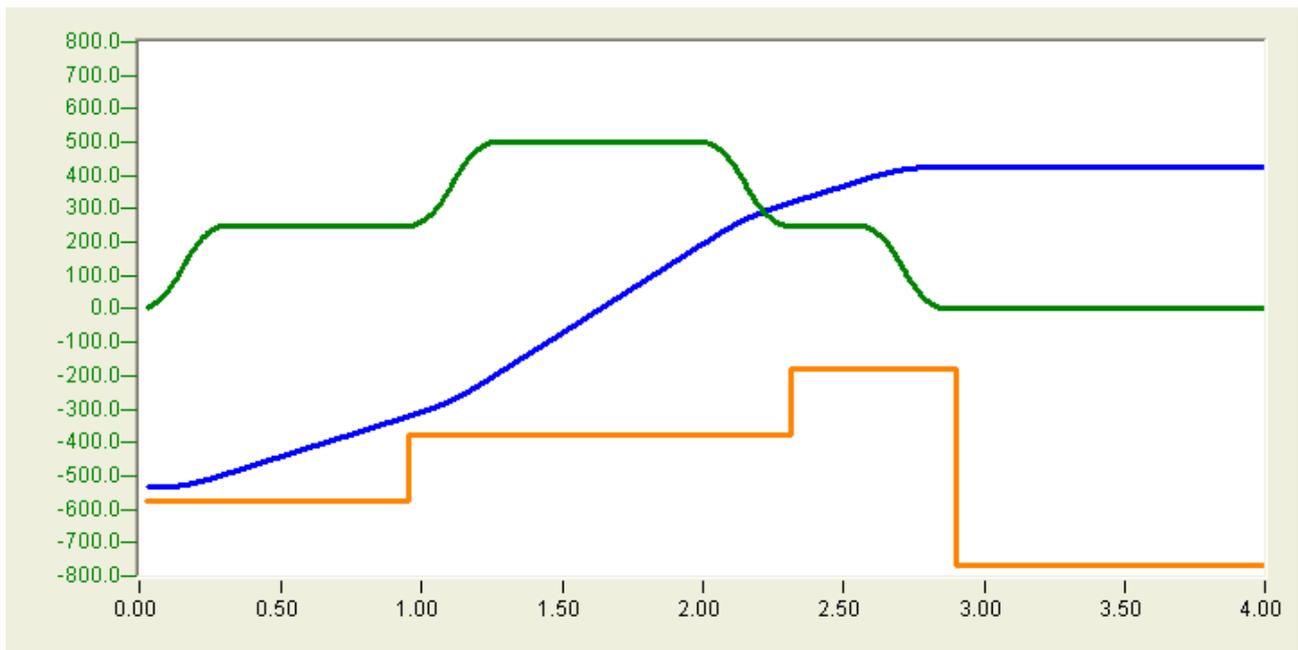
If the target velocity changes from a high velocity level to a lower level (N10 -> N20), the lower velocity will already have been reached at the start of the segment.

If the target velocity changes from a low velocity level to a higher level (N20 -> N30), the higher velocity is initiated with the segment transition. The system therefore always ensures that even at the segment boundary the current velocity does not exceed the programmed velocity.



green: Path velocity
 blue: Position
 orange: Block numbers

```
N10 G01 X600 F30000
N20 G01 X700 F15000
N30 G01 X900 F30000
M30
```



green: Path velocity
blue: Position
orange: Block numbers

```
N40 G01 X200 F15000  
N50 G01 X800 F30000  
N60 G01 X900 F15000  
M30
```

5.1.4 Smoothing of Segment Transitions

Overview

Segment transitions with no continuous second differential cause instability in the dynamics unless the path velocity is reduced to 0 at those points. For dynamically stable segment transition at a finite speed it is possible to smooth segment transitions with Bezier splines which alter the local geometry and ensure that the complete path has a continuous second differential.

Tolerance spheres

A tolerance sphere is laid around every segment transition within which the path may deviate from its pre-set geometry for smoothing purposes. The radius of the tolerance sphere ([parameterization \[► 319\]](#)) is predetermined by the user and applied modally for all segment transitions that imply no exact positioning or stop in the segment transitions. The radii of the tolerance spheres are automatically reset adaptively, thus preventing tolerance spheres from overlapping in the case of small segments.

Dynamic parameters

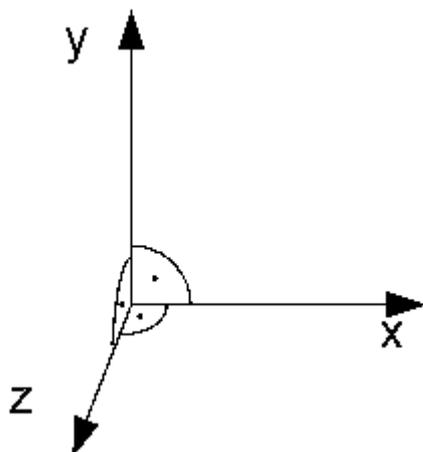
The smoothing enables faster dynamics. The system-determined maximum segment transition velocity *VeloLink* can be influenced by the user insofar as the system parameter C2 velocity reduction C2 ([parameterization \[► 319\]](#)) sets the segment transition velocity to $C2x \text{ VeloLink}$. The factor can be changed online.

General characteristics at segment transitions

When entering the tolerance sphere, the path acceleration is 0 and the path velocity equals the segment transition velocity. This is maintained within the tolerance sphere. The override is inactive within the tolerance sphere, i.e. the change of the velocity level caused by the override is interrupted within the tolerance sphere and continues after the exit from the tolerance sphere.

5.1.5 Co-ordinate System

The names of the axes of a machine tool are specified by DIN 66217. The letters X, Y and Z are allocated to the axes. These create a right-handed right-angle (Cartesian) coordinate system. In many machines, not all three axes are present at every location. In these cases individual letters are allocated in some meaningful way, and the axes that are not present are ignored.



5.1.6 Dimensional Notation

Dimensional data can optionally be referred to an absolute origin or to the current set value.

Absolute dimensions

Command	G90 (standard setting)
Cancellation	G91

All positional data in absolute dimensions are always given with reference to the currently valid origin.

In terms of tool movement, this means that, under absolute dimensioning, it is the position to which the tool should move that is described.

Incremental Dimensions

Command	G91
Cancellation	G90

When dimensions are incremental, positional data is related to the immediately preceding point. In addition to the path axes, the auxiliary axes (Q1..Q5) are also taken into account.

For the tool movement this means that the incremental dimension describes by how much the tool is moved.

Units

The units for length, angle etc. are described in the following table:

	Unit
Positions and lengths	mm
Angle	degree
Times	sec
Feed	mm/min

5.1.7 Working Plane and Feed Direction

In order to describe circles (except [CIP](#) [[▶ 135](#)]), and for the compensation of [cutter radius](#) [[▶ 183](#)] and [tool length](#) [[▶ 180](#)], it is necessary to specify the working plane.

Working Plane XY

Command	G17 (standard setting)
Cancellation	G18 or G19

The function G17 specifies the XY plane as the working plane and the feed direction as the Z direction.

The function acts as:

- Plane for [tool radius compensation](#) [[▶ 183](#)]
- Feed direction for [tool length compensation](#) [[▶ 180](#)] (offset)
- Plane for circle interpolation

● Changing the working plane

I The working plane cannot be changed while tool compensation is active.

Working Plane ZX

Command	G18
Cancellation	G17 or G19

The function G18 specifies the ZX plane as the working plane and the feed direction as the Y direction.

Working Plane YZ

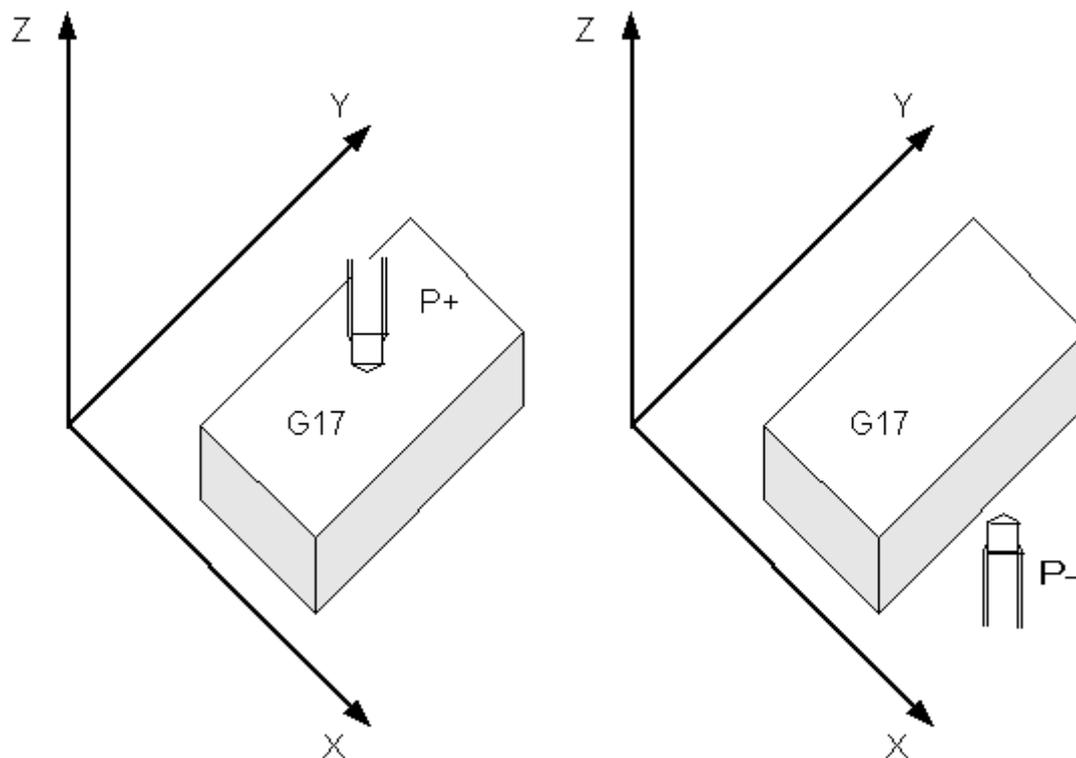
Command	G19
Cancellation	G17 or G18

The function G19 specifies the YZ plane as the working plane and the feed direction as the X direction.

Specification of the feed direction

Command	P
Parameter	+ feed direction positive (standard setting) - feed direction negative

Parameterization of the feed direction is required for tool length compensation. It is used to specify whether the tool operates above or below the workpiece.



Sample:

```
N10 G0 X0 Y0 Z0 F6000
N20 D2 P- Z
N30 G01 X100
N40 D0 Z
N50 M30
```

In this sample the length compensation operates below the workpiece.

5.1.8 Inch/metric dimensions

G70	Dimensions in inches
G71	Dimensions in millimeters (standard setting)
G700	Dimensions in inches with calculation of the feed
G710	Dimensions in millimeters with calculation of the feed

Dimensions in millimeters (G71) is active by default. Information on whether the coordinates have to be converted is stored in the machine parameters [▶ 14] (Interpreter tab). The basic dimension system in millimeters is also set there by default.

The effects of the changeover

If the basic dimension system is not the same as the current dimension system (set with G70 or G71), then certain parameters and co-ordinates must be converted. The conversion factor required here is stored in the machine parameters, like the basic dimension system. The changeover has effects on the following parameters:

- Path information for the path axes (X, Y & Z)
- Path information for the auxiliary axes (Q1..Q5)
- Intermediate point co-ordinates (I, J, K)
- Circle radii (B or U)
- Programmable zero shift

- Rounding radius (circle and spline smoothing)

There are also parameters that always remain in the **basic dimension system**, and are not converted. These include the

- adjustable zero shift
- Tool data
- feeds (except under G700 or G710)

Sample 1:

Basic dimension system: inch

```
...
N10 G71      (metric dimensions)
N20 G01 X100 (conversion is carried out)
N30 G70      (dimensions in inches)
N40 G01 Y100 (conversion is not necessary, because)
....        (the basic dimensions are also inches)
```

Sample 2:

Basic dimension system: millimeters

```
...
N10 G71 (metric dimensions)
N20 G01 X100 (conversion is not necessary, because)
      (the basic dimensions are also metric)
N30 G70 (dimensions in inches)
N40 G01 Y100 (conversion is carried out)
```

Zero shifts (NPV)

Adjustable zero shifts (G54-G57) always remain in the basic dimension system, and are not converted. In the case of the programmable zero shifts (G58 & G59) the effect depends on the current dimension system when the shift is selected.

Sample 3:

Basic dimension system: millimeters

```
...
N10 G71      (mm - default)
N20 G54      (activates adjustable zero offset shift)
N30 G58 X100 (programmable zero offset shift)
N40 G01 X0 F6000 (the axis travels to 100 in the machine co-ordinate system)
N50 G70      (inch)
N60 G01 X0    (zero offset shift is programmed under G71 => zero offset shift remains unchanged)
      (i.e. the axis does not move)
N70 G58 X100 (new programmable zero offset shift - now in inches)
N80 G01 X0    (axis moves out by zero offset shift - to 2540 in the machine co-ordinate system)
```

5.1.9 Single Block Operation

To test a new NC program, the NCI can be switched to single block mode with the function block [ltpSingleBlock](#) [► 241]. When single block mode is active, the NC program is stopped after each line. The user has to acknowledge execution of the next line. This can be done by pressing '**NC start (F5)**' in the XAE under the Editor tab or by setting the input 'bTriggerNext' in the PLC function block [ltpSingleBlock](#) [► 241].

A distinction is made between two modes:

- Interpreter single block mode
- NC kernel single block mode

The selection of the single block mode is not reset by an [ltpResetEx2](#) [► 231]. This means that if, for example, the NC kernel single block mode is active, it still remains active after a reset.

Interpreter single block mode

If interpreter single block mode is active, the NC program is stopped after each line in the **interpreter**. This remains true even if the line only contains calculations, and no physical movement is programmed. This enables re-writing of R-parameters, for example.

Interpreter single block mode should be activated before the NC program is started. If this is not possible, an M-function can be reserved for the activation and combined with a decoder stop.

If interpreter single block mode is activated during processing of the NC program without M-function and decoder stop, it is impossible to predict when it will be active. Theoretically it is possible that the memories in the NC kernel (SVB & SAF) are filled and contain more than 100 geometry entries. The single block can only take effect once these memories have been fully processed.

NC kernel single block mode

Like in interpreter single block mode, in NC kernel single block mode the NC blocks are executed individually. The difference is that in NC kernel single block mode all entries (e.g., geometry entries) have already passed through the interpreter. It is therefore not possible to overwrite R-parameters retrospectively, for example.

This operating mode has the advantage that single block mode can be enabled during processing of the NC program. If a geometry entry is executed (i.e. the axes are moved) during the activation, the system stops at the next possible end of segment. This is usually the current segment. For activation after program startup no M-function with decoder stop is required. However, an M-function with program end identifier (M02, M30) is required at the end of the program.

If NC kernel single block mode is used in conjunction with blending, block relaying takes place in the blending sphere. The programmed blending continues to be executed (from TwinCAT V2.10 Build 1301).

Alternatives to activation

We recommend activating single block mode with [ltpSingleBlock](#) [► 241].

For reasons of compatibility with previous TwinCAT versions, single block mode can be activated via the cyclic channel interface.

Single block mode can be selected or deselected in the cyclic channel interface of the PLC. To this end the variable 'nltpMode' has to be masked correctly in the PLC/NC channel interface.

Set bit 14 (0x4000) to switch on interpreter single block mode. Resetting the bit turns single block mode off again.

It is also possible to trigger the single block from the PLC by means of this interface. Bit 15 must be set for this. The effect is the same as activating NC start in the XAE.

5.1.10 Arithmetic Parameters

The arithmetic parameters (known as R-parameters for short) are interpreter variables that are named by an expression of the form "R<n>". Since 'n' is an integer in the range 0..999, a total of 1000 R-parameters are available. Of these, the first 900 values (R0..R899) are local variables for the NC channel. They can only be accessed by the channel's interpreter. The R-parameters R900..R999 are declared globally. They exist only once for each NC, and all channels access the same storage. This makes it possible to exchange data (e.g. for part tracing, collision avoidance etc.) over the channel boundaries.

Mathematical Calculations

The R-parameters (like the axis co-ordinates, feedrates etc.) are declared as variables of type 'double'. This makes full use of the computer's arithmetic capacity. The number of places before and after the decimal point is not restricted by a format specification. The arithmetical resolution does, nevertheless, have a limit. In practice this is only visible in particularly critical cases. Examples of this include the differences of very large numbers that are almost equal, or trigonometrical functions in particular ranges of angles.

Assignment of R-Parameters

```
N100 R5=17.5  
N110 R6=-4  
N120 R7=2.5 R8=1
```

As can be seen in the third line, it is quite possible to make more than one assignment in one block. This speeds interpretation slightly, but it can be more difficult to localize an error in the line.

Calculation formula

A calculation formula is an extension of assignment. It consists of a target parameter, an assignment sign and a series of values (R-parameters and constants) separated by arithmetical instructions.

```
N100 R1=R2+R3-17.5*R9/2.5
```

Such a formula, contrary to normal mathematical practice, is processed strictly from left to right.

The illustrated formula is calculated as follows:

1. The contents of R2 is loaded into the arithmetic unit
2. The contents of R3 is loaded into the arithmetic unit
3. The arithmetic unit carries out the + instruction
4. The value 17.5 is loaded into the arithmetic unit
5. The arithmetic unit carries out the - instruction
6. The contents of R9 is loaded into the arithmetic unit
7. The arithmetic unit carries out the * instruction
8. The value 2.5 is loaded into the arithmetic unit
9. The arithmetic unit carries out the / instruction
10. The content of the arithmetic unit is stored in R-parameter R1

Mathematical functions

The interpreter provides standard computing functions. DIN 66025 does not specify any syntax here. The computing functions are called via @6xx (see appendix - [@-command overview \[► 193\]](#)).

The trigonometrical functions are always calculated in degrees.

Sample:

```
N10 R2=0 R3=45  
N20 @630 R2 R3
```

In this sample the sine of R3 is calculated in degrees. The result is then written into R2.

R-parameter access from the PLC

You can read the R-parameters into the PLC, or write the R-parameters from the PLC. Special PLC function blocks are provided for this purpose

- [ItpReadRParams \[► 276\]](#)
- [ItpWriteRParams \[► 289\]](#)

During writing of the R-parameters, ensure that the interpreter is ahead of the block execution. In other words, writing of the R-parameters from the PLC should take place before the NC program starts or be linked to a [decoder stop \[► 166\]](#).

For debugging purposes, all R-parameters can be written to a file at any time. This process can be triggered via ADS (see ADS interface - channel functions IndexOffset 0x24 & 0x25).

Other functions

RToDwordGetBit

This function converts an R-parameter to a DWord and then checks whether a particular bit is set. The result is again stored in an R-parameter.

Command	RToDwordGetBit[<dest>; <src>; <bit>]
Parameter <dest>	R-parameter in which the result is entered
Parameter <src>	R-parameter containing the number that is to be converted and checked
Parameter <bit>	Bit to be checked (0..31)

Sample:

```
N10 R1=7
N20 RToDwordGetBit[R2;R1;0]
R10=31
N30 RToDwordGetBit[R3;R1;R10]
```

Enter 1 in R2 and 0 in R3.

Initialization of R-parameters

'set RParam' is used to assign a value to a contiguous block of R-parameters.

Command	#set RParam(<start index>; <count>; <value>)#
Parameter <start index>	Describes the first R-parameter to be written
Parameter <count>	Number of R-parameters to be written
Parameter <value>	Assigned value

Sample:

```
N10 G01 X100 Y200 F6000
N15 R2=3000
N20 #set RParam( 1; 2; 0.0 )# (R2 is overwritten again here)
N30 G01 X500
```

Saving R-Parameters

If the content of [R-parameters](#) [► 130] is required for subsequent use, while in the meantime the R-parameters are used for a different purpose, it can temporarily be stored in the values stack of the arithmetic unit.

Two possibilities exist for this:

- enumeration of the R-parameters
- giving the range of R-parameters

Saving the values:

Command	@40 <number> R<n> R<m>... @41 <1st R-parameter> <last R-parameter>
---------	---

Restoring the values:

Command	@42 <number> R<n> R<m> @43 <last R-parameter> <1st R-parameter>
---------	--

When restoring the values, call the parameters in reverse order.

Sample 1:

```
(saving the data)
N100 @40 K4 R800 R810 R823 R4
```

```
N110 R800=4711
N120 ...

(restoring the data)
N200 @42 K4 R4 R823 R810 R800
```

Sample 2:

```
(saving the data)
N100 @41 R800 R805

N110 R800=4711
N120 ...

(restoring the data)
N200 @43 R805 R800
```

Stack size

The value stack of the arithmetic unit has limited capacity. If it overflows, the NC program is interrupted with an error message. That can occur as the value is saved, but can also occur in the course of subsequent formula evaluation.

5.2 Programming Movement Statements

5.2.1 Referencing

By default, axis referencing (homing) should take place before the 3D-group is formed from the PTP channel. Or it can be done from the NC program.

If axes are referenced in PTP mode, it can be done for several axes simultaneously. If axes are referenced from the NC program, it can only be done for one axis at a time.

Command	G74
Cancellation	End of block

Sample:

```
N10 G74 X
N20 G74 Y
```

Referencing with own block

Referencing must be carried out within its own block. G74 may only refer to one axis. This command is only applicable for the main axes (X,Y,Z).

5.2.2 Rapid Traverse

Command	G0
Cancellation	G1 [▶ 134], G2 [▶ 135] or G3 [▶ 135]

Rapid traverse is used to position the tool quickly, and is not to be used for machining the workpiece. With G0 the axes are moved with linear interpolation as fast as possible. The velocity is calculated with MIN (Rapid Traverse Velocity (G0), Reference Velocity, Maximum Velocity).

If a number of axes are to be driven in rapid traverse, the velocity is determined by that axis that requires the most time for its movement.

An accurate stop ([G60 ▶ 138](#)) is cancelled with G0.

The rapid traverse velocity is set individually for each axis. This can be edited in the axis parameters in the XAE under NCI parameters.



General Settings Parameter Dynamics Online Functions Coupling Compensation					
	Parameter	Offline Value	Online Value	Type	Unit
+	Maximum Dynamics:				
+	Default Dynamics:				
+	Manual Motion and Homing:				
+	Fast Axis Stop:				
+	Limit Switches:				
+	Monitoring:				
+	Setpoint Generator:				
-	NCI Parameter:				
	Rapid Traverse Velocity (G0)	2000.0		F	mm/s
	Velo Jump Factor	0.0		F	
	Tolerance ball auxiliary axis	0.0		F	
	Max. position deviation, aux. axis	0.0		F	
+	Other Settings:				

5.2.3 Linear Interpolation

Command	G1 or G01 (standard setting)
Cancellation	G0 [▶ 133], G2 [▶ 135] or G3 [▶ 135]

Under linear interpolation the tool moves, with feedrate F, along a straight line that can be freely located in space. The movement of the axes involved is completed at the same moment.

The feedrate (short: feed), F, describes the rate of displacement in millimeters per minute. This value is effective globally, so that it is not necessary to program it again if the same feed is to be used later for other geometrical movements.

Sample:

```
N10 G90
N20 G01 X100.1 Y200 F6000
```

In this example the axes are moved linearly to the position described. The Z axis is not mentioned in this program, and therefore retains its old position.

5.2.4 Circular Interpolation

Circular arcs can be programmed in a number of ways. Two types must be distinguished. One of these is an arc in the working plane [▶ 126] (e.g. the XY plane), and the other is an arc that can be freely located in space (a CIP circle).

Clockwise circular interpolation

Command	G2 or G02
Cancellation	G0 [▶ 133], G1 [▶ 134] or G3 [▶ 136]

Function G2 describes the path of a circular arc clockwise. This function requires the working plane [▶ 126] to have already been defined (G17 [▶ 126] by default).

In order to describe the circle unambiguously, further parameters are required in addition to the end point. A choice is available between center point programming and radius programming.

Radius programming

In radius programming the radius of the circle is programmed as well as the end point. Either of the letters 'B' or 'U' may be used for the radius.

Since the direction is prescribed with G2, the circle is also unambiguously described. The starting point is determined by the foregoing geometrical movements.

Sample 1:

```
N10 G01 G17 X100 Y100 F6000
N20 G02 X200 B200
```

● Angle programming for angles >180°

I If an angle of more than 180° is to be traversed, the radius must be stated negatively.

● Full circle programming

I The start and the end points must be different, so that the center can be calculated. Radius programming can therefore not be used for programming a full circle. Centre point programming can be used for this purpose.

Centre point programming

Centre point programming represents an alternative to the method that has just been described. The advantage of center point programming is that full circles can also be described in this way.

Under the standard settings, the center point is always given relatively to the starting point of the arc. The parameters I, J and K are used for this purpose. With

- I for the X-component
- J for the Y-component and
- K for the Z-component.

At least one of these parameters is 0 and therefore does not have to be included in the program.

Sample 2:

```
N10 G01 G17 X100 Y100 F6000
N20 G02 I50 J0 (J is optional) X200
N30 M30 (program end)
```

Sample 3:

```
N10 G01 G18 X100 Y100 Z100 F6000
N20 G02 I0 K50 X150 Z150 (quarter circle in ZX plane)
N30 M30
```

By programming an item of machine data it is however also possible to enter the center point absolutely. The command @402 is required for write access to a machine data bit.

In the following example, the circle from the first example is programmed using the absolute circle center.

Sample 4:

```
N10 G01 G17 X100 Y100 F6000
N20 @402 K5003 K5 K1 (centre point programming on)
N30 G02 I150 J100 X200
N40 @402 K5003 K5 K0 (centre point programming off)
N50 M30
```

Anticlockwise Circular Interpolation

Command	G3 or G03
Cancellation	G0 [▶ 133], G1 [▶ 134] or G2 [▶ 135]

The function G3 describes a circular arc anticlockwise. The parameters and entry possibilities are the same as under G2.

Circular accuracy

Command	#set paramRadiusPrec(<param>)#
Parameter	param: Maximum permitted radius tolerance 0.001 < param < 1.0 (default 0.1)

The 'set paramRadiusPrec' function is used to parameterize the required circular accuracy. This parameter affects circles programmed with G02 or G03.

With center point programming, an error is generated if the difference in radius length is greater than <param>.

Centre point correction

Command	CPCON (standard setting)
Cancellation	CPCOF

In center point programming the circle is overdetermined. For data consistency, the center point is usually corrected. Normally only a marginal modification of the center point is required. After the center point correction, the magnitude of the input radius equals the output radius.

If the start and end point are very close together, the center point offset may be large. This may lead to problems with automatically generated G-Code (postprocessor). For manually written G-Code, the CPCON setting (center point correction on) is recommended.

CIP circle

Command	CIP
Cancellation	End of block

The circles discussed so far can only be used in the principal planes. The CIP circle can also be used to program a circle anywhere in space. For this purpose it is necessary to program not only an end point but also some other point on the path.

So that the circle can be described unambiguously, it is necessary that the three points (the starting point is given implicitly) must not be collinear. It is thus not possible to program a full circle in this way.

I, J and K are available as path point parameters. By default their values are relative to the starting point of a circular path.

Sample 5:

```
N10 G01 X100 Y100 F6000
N20 CIP X200 Y200 I50 J50 K50
```

Note For the CIP circle motion, [cutter radius compensation \[▶ 183\]](#) **must not be active.**

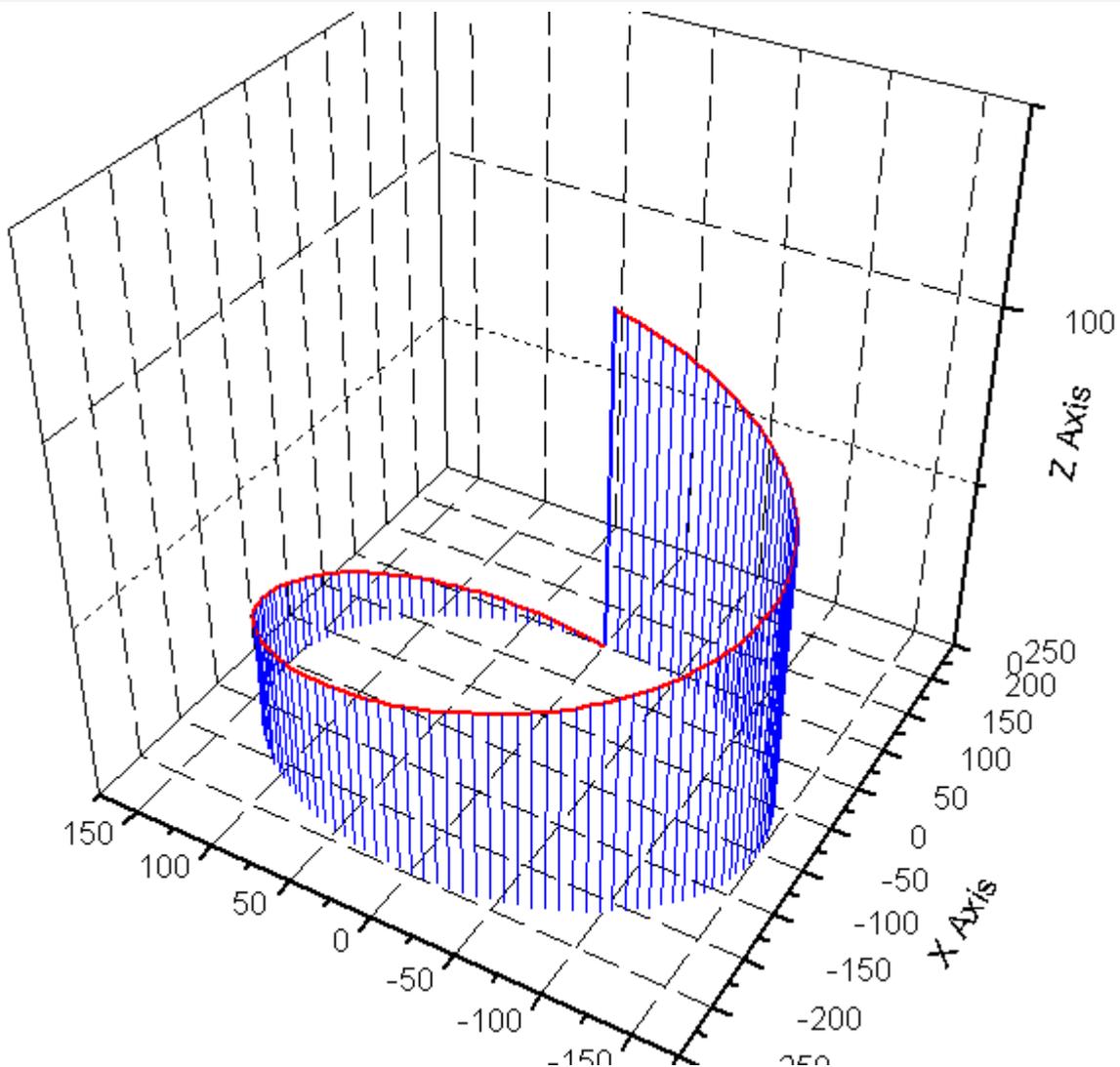
5.2.5 Helix

If a circular motion is superimposed onto a perpendicular linear movement, a helix is obtained. A helix can only be programmed in the principal planes. The same parameters as are used for a circle in the principal plane are used. At the same time the axis that is perpendicular to the plane is driven.

The helix can be used together with the [cutter radius compensation \[▶ 183\]](#).

Sample:

```
N10 G01 G17 X100 Y0 Z0 F6000
N20 G03 I-50 Z100
M30
```



5.2.6 Dwell Time

Command	G4 or G04
Cancellation	End of block
Parameter	F or X

G4 is used to switch on a dwell time. It is used to interrupt workpiece machining between two NC blocks for a programmed time (in seconds).

Sample:

```
N10 G01 X100 F6000
N20 G04 X0.5 (pause in sec)
N30 G02 X300
...
```

Note The dwell time must be programmed in a dedicated set, and the parameters (X or F) must be programmed after G04.

5.2.7 Accurate Stop

block-by-block

Command	G9 or G09
Cancellation	End of block

The accurate stop instruction is used, for example, when sharp contour corners must be manufactured. At the contour transition the set path velocity is reduced to zero and then increased again. This ensures that the programmed position is approached precisely.

Note G09 acts only on the set value side. The actual values can be checked with TPM (target position monitoring), for example.

modal

Command	G60
Cancellation	G0 [▶ 133]

Description:

see above

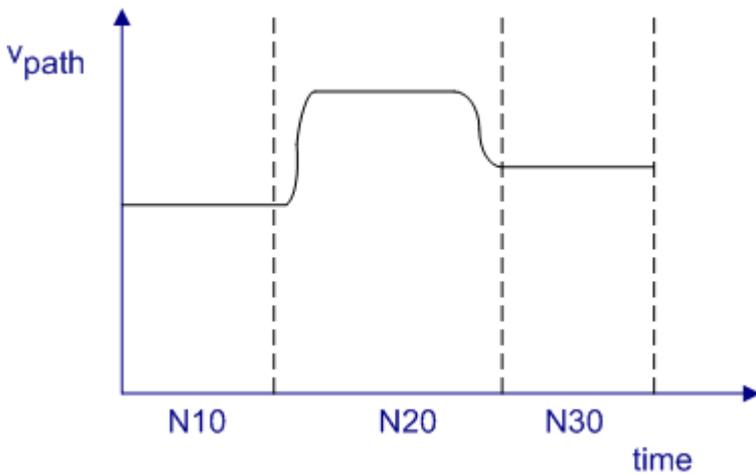
see also [target position monitoring \[\[▶ 142\]\(#\)\]](#) (TPM)

5.2.8 Feed interpolation

Constant feed interpolation

Command	FCONST (standard setting)
Cancellation	FLIN

The programmed velocity is applied as fast as possible with the constant feed interpolation (default).



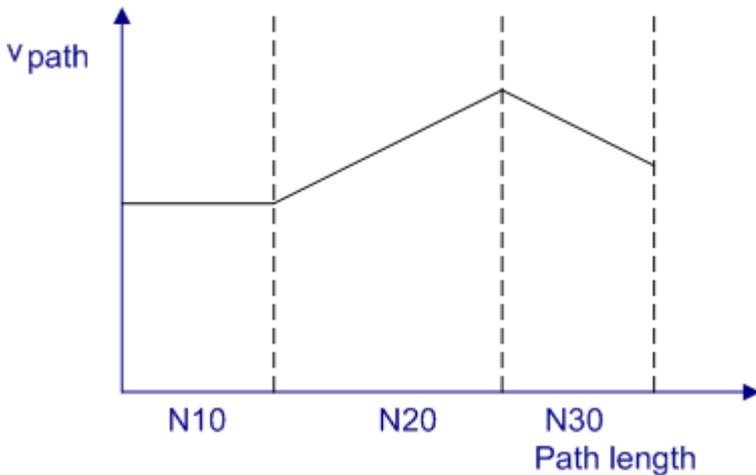
Sample 1:

```
N05 FCONST
N10 G01 X1000 F50000
N20 G01 X2500 F80000
N30 G01 X3500 F60000
...
```

Linear feed interpolation

Command	FLIN
Cancellation	FCONST

The linear feed interpolation transfers the velocity linearly over the path from v_{start} to v_{end} .



Sample 1:

```
N05 FCONST
N10 G01 X1000 F50000
N15 FLIN
N20 G01 X2500 F80000
N30 G01 X3500 F60000
...
```

Note If the velocity on the segment transition has to be reduced more drastically than the programmed segment velocity, due to the geometry or M function for example, then the linear velocity is maintained as long as possible. The reduced segment velocity will be delayed, only if required dynamically.

5.2.9 Zero Offset Shifts

A range of zero offset shifts are available in TwinCAT NC I. They describe the distance between the origins of the workpiece and of the machine.

Zero shift suppression

Command	G53 (standard setting)
Cancellation	G54 [▶ 140] to G59 [▶ 141]

The zero shift is suppressed modally with G53. The suppression affects both the adjustable and the programmable zero shift.

Adjustable zero shift

Command	G54 G55 G56 G57
Cancellation	G53 [▶ 140] or selection of another configurable zero shift

The commands G54 to G57 can be used within the NC program to switch back and forth between the zero shifts.

Parameterization

The configurable zero shift can be parameterized in different ways

1. PLC function block `ItpWriteZeroShiftEx` [▶ 247] (recommended standard)
2. XAE `Interpreter element` [▶ 14]
3. from the DIN-program

The parameters are saved for each interpolation channel. This means that the adjustable zero shifts are channel dependent.

Note The selection of a zero shift must be made in its own block. In order for the movement corresponding to the shift to be actually made it is necessary that at least the axes involved are named in a following geometrical block.

Sample 1:

```
N10 G01 X100 Y0 Z0 F6000
N20 G54 (activates adjustable zero offset shift (NPV))
N30 G01 X Y Z
N40 M30
```

In sample 1 all involved axes are named in line 30. The effect of this is that the zero shifts are applied to all the axes.

Sample 2:

```
N10 G01 X100 Y0 Z0 F6000
N20 G54 (activates adjustable zero offset shift (NPV))
N30 G01 X200 Y
```

In line 30 of sample 2 the X axis is taken to position 200 + shift in the X direction. The Y axis only moves to accommodate the shift, and the Z axis is not moved.

Parameterization from the DIN program

Command	<code>#set paramZeroShift(G<n>; <value x>; <value y>; <value z>)#</code>
Parameter G<n>	Zero shift to be parameterized (G54..G59)
Parameter <value>	Coordinates of the zero shift

`'#set paramZeroShift(..)#'` parameterizes the zero shift but does not activate it. This requires explicit programming of the G-Code.

Sample 3:

```
N10 G01 X100 Y0 Z0 F6000
N20 R12=200
N30 #set paramZeroShift( G54; 100.0; R12; -20)#
N40 G54 (activates adjustable zero offset shift (NPV))
N50 G01 X200 Y Z
```

Programmable zero shift

Command	G58 or G59
Cancellation	G53 ▶ 140

Programmable zero shifts exist in addition to the adjustable ones. This type of zero shift is directly described from the NC program.

● Addition of zero shifts

I The programmable zero shift is only effective when the adjustable zero shift is active. This means that the total shift is the sum of

- set zero shift (G54, G55, G56 or G57)
- first programmable zero shift (G58)
- second programmable zero shift (G59)

Sample 4:

```
N10 G01 X100 Y0 Z0 F6000
N20 G54 (activates adjustable zero offset shift (NPV))
N30 G58 X0.5 Y0.5 Z0.5 (1st prg. zero offset shift)
N50 X Y Z (movements for the zero offset shift)
...
M30
```

Behavior with incremental dimension notation

Default behavior

Changing the origin also affects the incremental dimension.

Sample 5:

```
N10 G01 X100 Y0 Z0 F6000
N20 G54 (activates adjustable zero offset shift (NPV))
N25 G58 X10 Y10 Z0
N30 G91 (Incr. dimensions)
N40 G01 X200 Y0
N50 ...
```

In N40 Y moves to 10 in the basic coordinate system. A shift in origin also shifts the point of reference for incremental dimension programming, resulting in a travel path for Y.

In this way a contour, which is fully programmed based on the incremental dimension, can be positioned at any point through a zero shift.

The behavior of G91 is parameterizable.

Command	Description
ZeroShiftIncOn	The zero shifts are also applied under G91 once the axis is named. (standard setting)
ZeroShiftIncOff	The zero shift is not applied under G91.

Sample 6:

```
N10 G01 X100 Y0 Z0 F6000
N15 ZeroShiftIncOff
N20 G54 (activates adjustable zero offset shift (NPV))
N25 G58 X10 Y10 Z0
N30 G91 (Incr. dimensions)
N40 G01 X200 Y
N50 ...
```

Since 'ZeroShiftIncOff' is set in sample 6 , the X-axis in N40 is moved by 200 mm independently of the new zero shift. The Y-axis does not move as no target coordinate is programmed for it.

See also [ToolOffsetIncOn/Off \[► 180\]](#)

5.2.10 Target Position Monitoring

Command	TPM
Cancellation	End of block

The command 'TPM' is used to trigger target position monitoring from the NC program. At the end of the geometry this always leads to an accurate stop on the set value side and subsequent checking of the target position window. Block relaying takes place when the monitoring conditions are met for all axes in the group.

Like for PTP, this function is enabled and parameterized individually for each axis. This means that different limits can be selected for auxiliary axes than for the path axes, for example.

Sample 1:

```
N10 G01 X100 Y100 F6000
N20 G01 X300 Y100 TPM
...
```

At the end of the motion of N20, target position monitoring is performed both for the X axis and for Y axis (provided target position monitoring is enabled for both axes).

Sample 2:

```
N10 G01 X100 Y100 F6000
N20 G01 X300 Y100
N30 M61 (Type Handshake)
N40 TPM
...
```

TPM can also be programmed in a dedicated block. In this case the last positioning is checked (of N20 in this case).

General Settings Parameter Dynamics Online Functions Coupling Compensation					
Parameter	Offline Value	Online Va...	T...	Unit	
+ Maximum Dynamics:					
+ Default Dynamics:					
+ Manual Motion and Homing:					
+ Fast Axis Stop:					
+ Limit Switches:					
- Monitoring:					
Position Lag Monitoring	TRUE	TRUE	B		
Maximum Position Lag Value	5.0	5.0	F	mm	
Maximum Position Lag Filter Time	0.02	0.02	F	s	
Position Range Monitoring	TRUE	TRUE	B		
Position Range Window	5.0	5.0	F	mm	
Target Position Monitoring	TRUE	TRUE	B		
Target Position Window	2.0	2.0	F	mm	
Target Position Monitoring Time	0.02	0.02	F	s	
In-Target Alarm	TRUE	TRUE	B		
In-Target Timeout	5.0	5.0	F	s	
Motion Monitoring	FALSE	FALSE	B		
Motion Monitoring Window	0.1	0.1	F	mm	
Motion Monitoring Time	0.5	0.5	F	s	
+ Setpoint Generator:					
+ NCI Parameter:					
+ Other Settings:					

Download Upload Expand All Collaps All Select All

Note If target position monitoring is enabled for an axis, the target position alarm (PEH) should also be active. Time monitoring results in a channel error after the timeout (or before), if the axis is not yet in the target position window. In order to avoid unnecessary channel errors, a sufficiently large timeout value should be selected (e.g. 5 - 10 s). If no PEH time monitoring is active and the axis is permanently outside the position window, no block relaying takes place and the NC remains stationary when viewed from outside. The SAF is in Waiting state (not to be confused with Interpreter state).

See also [accurate stop \[▶ 138\] \(G09\)](#).

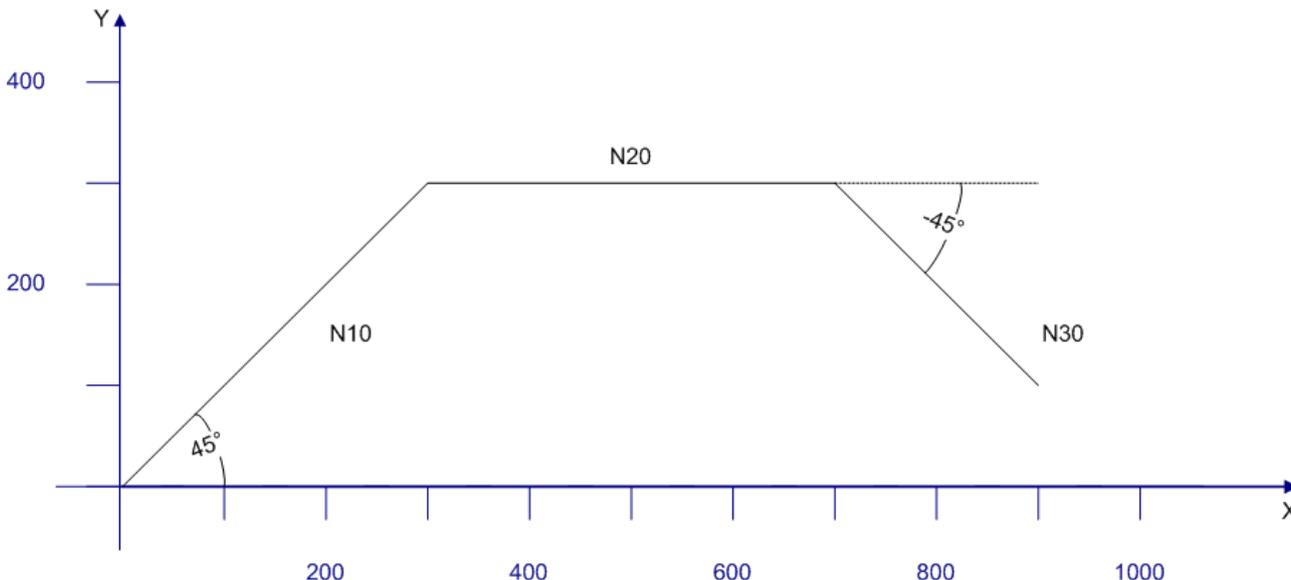
5.2.11 Contour definitions

Angle and segment length

In this type of programming the angle and the magnitude (segment length) are always quoted, similarly to polar co-ordinates.

Parameter	Description
ANG	Angle in degrees with reference to the abscissa ($-360 \leq \text{ang} \leq 360$)
SEG	Magnitude of the segment length

Sample 1:



```
N10 G01 ANG=45 SEG=424.264 F60000
N20 G01 ANG=0 SEG=400
N30 G01 ANG=-45 SEG=282.843
```

or

```
N10 G01 ANG=45 SEG=424.264 F60000
N20 G01 X700 Y300
N30 G01 ANG=-45 SEG=282.843
```

Restrictions:

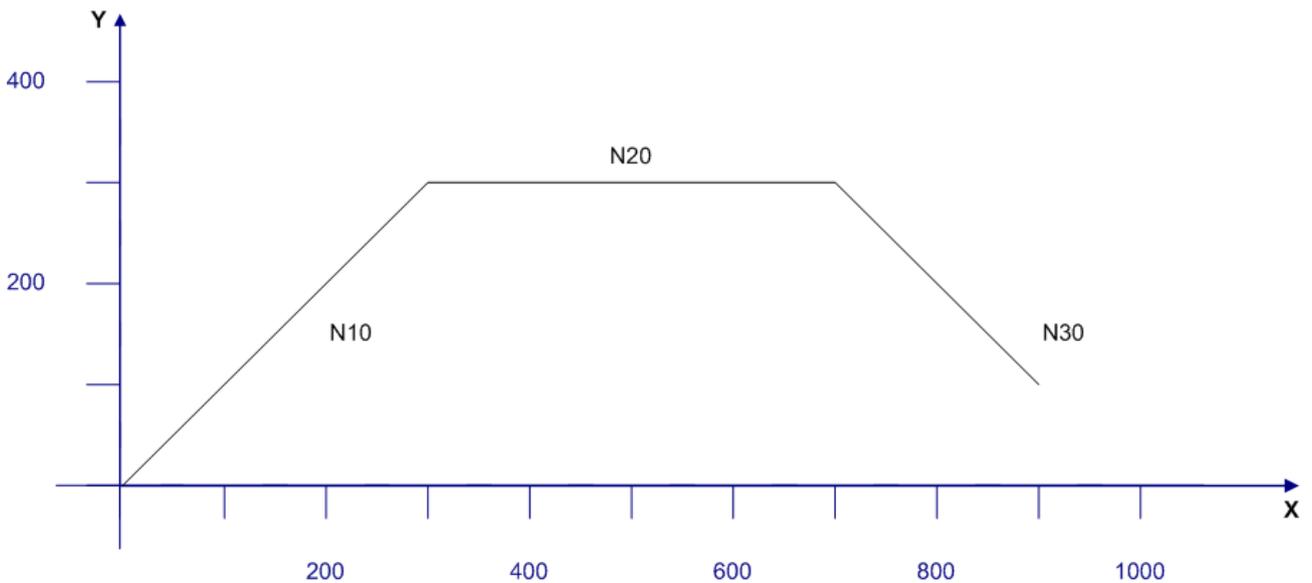
- The programming may only be done in the chosen principal plane.
- The length of the segment must be greater than zero, and refers to the projection in the principal plane.

Note It is additionally possible to program rounding or chamfering. The ANG and SEG parameters must be programmed in every block. The assignment may use R-parameters, but formulas cannot be programmed.

Angle and one component in the plane

As above, an angle is programmed, but the length of the segment is no longer specified directly. It is calculated from a component in the selected principal plane.

Sample 2:



```
N10 G01 ANG=45 X300
N20 G01 ANG=0 Z700
R10=100
N30 G01 ANG=315
X=R10
```



Runtime error

If either two components in the plane are quoted or none at all, the result is a runtime error. A runtime error is also generated if the movement is parallel to the abscissa or to the ordinate, and there is therefore no intersection.

5.2.12 Rotation

It is also possible to program a rotation as well as the zero shift [▶ 139]. A distinction is drawn between absolute and additive rotation.

The rotation can turn the co-ordinate axes (X, Y and Z) in the workpiece coordinate system.

This makes it possible to machine inclined surfaces (in the plane or in space).

Absolute Rotation

Command	ROT X<value(x)> Y<value(y)> Z<value(z)>
Cancellation	ROT (without parameters)

The rotation instructions must be programmed in their own block. Angles must always be stated in degrees.

Direction of Rotation

A positive angle describes rotation in the direction of the positive co-ordinate axis, the rotation being anti-clockwise.

Carrying Out the Rotation

The sequence of rotations is of critical importance when a coordinate system is being rotated. In TwinCAT NC I rotations are always carried out in the following sequence around the global coordinate system:

1. Rotation around the Z-axis,
2. Rotation around the Y-axis,
3. Rotation around the X-axis.

This sequence is maintained even if the parameters are programmed in a different order.

The origin of the tool coordinate system is always used as the center point of the rotation. This means that the total zero offset shift currently active describes the rotation center.

Additive Rotation

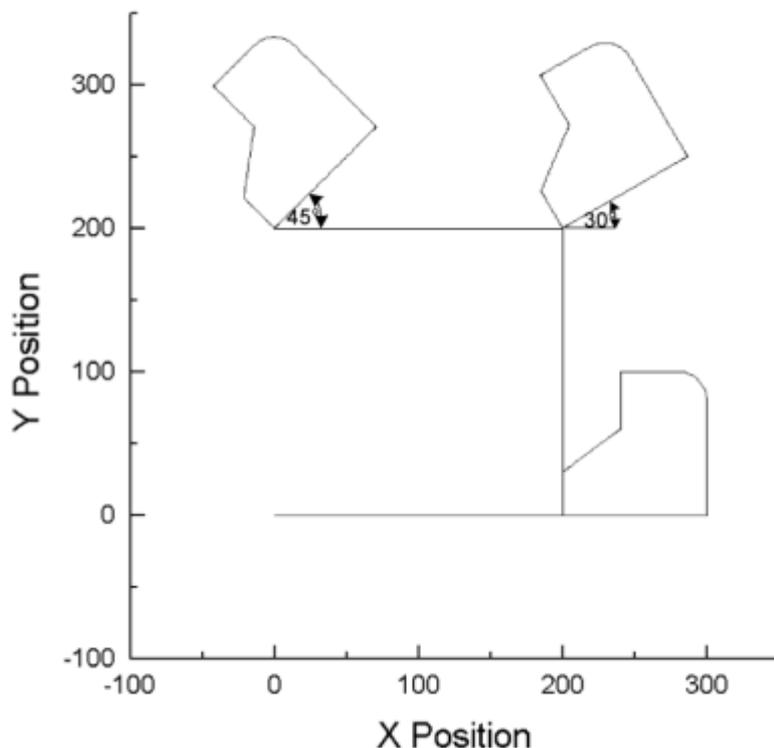
In addition to absolute programming of rotation it is also possible to carry this out additively. The same conditions apply to this as do to absolute rotation.

Command	AROT X <Wert(x)> Y<Wert(y)> Z<Wert(z)>
Cancellation	ROT (without parameters)

Sample:

```
N10 G01 G17 X0 Y0 Z0 F60000
N20 G55
N30 G58 X200 Y0
N50 L47
N60 G58 X200 Y200
N65 ROT Z30
N70 L47
N80 G58 X0 Y200
N90 AROT Z15
N100 L47
N110 M30
```

```
L47
N47000 G01 X0 Y0 Z0 (movements for zero shift & rotation)
N47010 G91 (incremental dimensions)
N47020 G01 X100
N47030 G01 Y80
N47040 G03 X-20 Y20 I-20 J0
N47050 G01 X-40
N47060 G01 Y-40
N47070 G01 X-40 Y-30
N47080 G01 Y-30
N47090 G90
N47100 M17
```



In this example, the same contour is traversed under different rotations. Since the contour (L47) is programmed in incremental dimensions, and the starting point is described by means of the programmed zero shift, the rotation is clear to see.

Note:

Once the ROT or AROT command has been programmed, the complete path vector (X, Y & Z) must be assigned.

Rotation extensions

In the default configuration the whole path vector must be programmed after each ROT command. Since this is difficult to realize in some applications, this calculation can optionally be performed automatically in the interpreter. To use this option, 'RotExOn' should be included at the start of the NC program.

Command	RotExOn
Cancellation	RotExOff

Sample:

```
N10 RotExOn
...
N100 G54 (activate zero point & point of rotation)
N110 ROT X90
N120 G0 Z3 (preposition the tool)
N130 G01 Z-10 F6000 (lower to cutting depth)
N140 G01 X100
N150 G01 Z3 (raise to preposition)
...
N1000 RotExOff
N1010 M30
```

Calculate rotation

Command	CalcRot[R<s>; R<t>; R<u>]
	CalcInvRot[R<s>; R<t>; R<u>]
Parameter	The 3 R-parameters describe the vector to be calculated. The calculation will write the result into this R-parameter, and the original value will therefore be overwritten.

The function **CalcRot** rotates a three-dimensional vector through the current rotation angle. The rotation angles had been determined by ROT or AROT. The sequence of the calculation is the same as is used for the rotation itself, that is Z, Y and X.

The **CalcInvRot** function behaves in precisely the opposite way. The signs of the currently valid rotation angles are inverted, and the order of calculation is X, Y and Z. In other words, the vector is turned back, so to speak.

Neither CalcRot nor CalcInvRot generate any geometry, but merely carry out the calculation of the vector.

Sample:

```
N10 G01 X40 Y10 Z0 F6000 (the axes are moved
without rotation)
N20 R1=40 R2=10 R3=0

N30 ROT Z45

(What is the position to which X, Y, must be taken so that no
movement is executed?)
N40 CalcInvRot[R1; R2; R3]
N50 G01 X=R1 Y=R2 Z=R3 (R1=35.35 R2=-21.21 R3=0)
N60 ...
```

Command	RotVec[R<x>; R<y>; R<z>; R<α>; R<β>; R<γ>]
Parameter	The 3 R-parameters (x..z) describe the vector to be rotated through. The calculation will write the result into this R-parameter, and the original value will therefore be overwritten. The last 3 R-parameters describe the angle.

The function **RotVec** rotates a three-dimensional vector through the specified angle. The order of the rotation is Z, Y and X, like for ROT. RotVec is a calculation routine solely for rotating a vector. It has no effect on ROT or AROT.

5.2.13 Mirror

The mirror functionality changes the sign of named axes. This enables subroutines to be reused.

Mirroring

Command	Mirror <opt. X> <opt. Y> <opt. Z>
Cancellation	Mirror (without parameters)

The mirror instructions must be programmed in a dedicated block. Mirrored axes must be named without further parameters.

Sample:

```

N20 G54
N30 G58 X100 Y100
N40 L100

N50 G58 X-100 Y100
N60 Mirror X
N70 L100

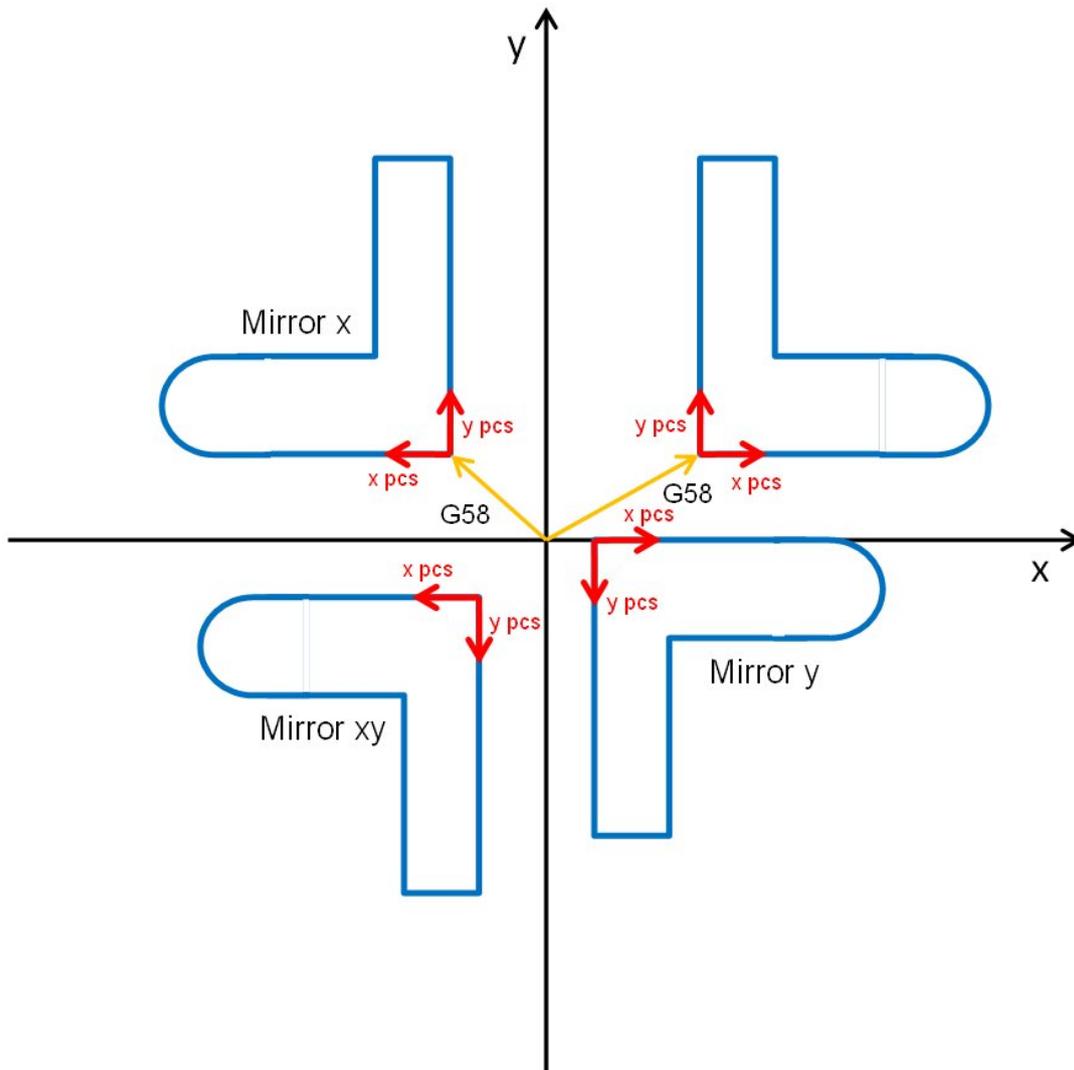
N80 G58 X-50 Y-50
N90 Mirror X Y
N100 L100

N110 G58 X10 Y-10
N120 Mirror Y
N130 L100

N140 Mirror (turn off mirror)
N150 G0 X0 y0
M02

L100
N1000 G0 X200 Y0 Z10 F60000 (move to start pos)
N1020 G01 Z0
N1030 G03 X200 Y100 J50
N1040 G01 X50
N1050 G01 Y400
N1060 G01 X0
N1070 G01 Y0
N1080 G01 X200
N1090 G01 Z10
M17

```



If a zero shift is present (G54...G59), the mirror functionality depends on the currently programmed coordinate system.

5.2.14 Smoothing of segment transitions

5.2.14.1 Overview

Overview

In general, at segment transitions polygon functions (G01 blocks) contain kinks within their contour. At these transitions polygon functions are not steadily differentiable with respect to their spatial coordinate, thus leading to dynamic-unsteadinesses, if at these transitions the path velocity is not reduced to zero value. To actually avoid to have to reduce path velocity to zero value segment transitions of polygon functions can be smoothed out by blending at those transitions.

	Execution	Supported Segment Transitions	Acceleration of Axis Components	Max. Tolerance	Adaptive Tolerance Radius	Command
Circular Smoothing [▶ 154]	Interpreter	Straight line/ straight line	Step change in acceleration (value parameterizable via the C1 factor)	1/2 of the input or output segment	No	paramCircularSmoothing (...)
Parabolic Smoothing [▶ 151] <type>: 2	NC kernel	Straight line/ straight line	Step change in acceleration to a constant level (value parameterizable via the C1 factor)	1/3 of the input or output segment	Can be selected	paramVertexSmoothing (...)
Biquadratic Smoothing [▶ 151] <type>: 3	NC kernel	Straight line/ straight line	Constant acceleration - the acceleration is 0 at the entry and exit - no intermediate point required	1/3 of the input or output segment	Can be selected	paramVertexSmoothing (...)
Bézier Curve of the 3rd Order [▶ 151] <type>: 4	NC kernel	All	Step change in acceleration to a linear level (can be parameterized with the C1 factor)	1/3 of the input or output segment	Can be selected, has an effect for straight-line transitions	paramVertexSmoothing (...)
Bézier Curve of the 5th Order [▶ 152] <type>: 5	NC kernel	All	Constant acceleration - the acceleration is 0 at the entry and exit - no intermediate point required	1/3 of the input or output segment	Can be selected, has an effect for straight-line transitions	paramVertexSmoothing (...)
'Old' Bézier Blending [▶ 152] <type>: 1	NC kernel	All	Constant acceleration - the acceleration is 0 at the entry, the exit and at the symmetric intermediate point	1/4 of the input or output segment	No	paramSplineSmoothing (...) paramVertexSmoothing (...)

Blending takes effect from the transition between the subsequent two segments.



Principle of Blending

The radius of the tolerance sphere can be altered at any time within the NC program and can be switched off again by setting the radius to 0. Blending remains active until the next reset of the interpreter or a TwinCAT runtime restart.

5.2.14.2 Parabolic smoothing

Parabola smoothing

Command	#set paramVertexSmoothing(<type>; <subtype>; <radius>)#
Parameter <type>	For parabola smoothing: 2
Parameter <subtype>	1: Constant tolerance radius [▶ 154] 2: Distance between intersection and vertex [▶ 154] 3: Adaptive tolerance radius [▶ 154]
Parameter <radius>	Max. radius of the tolerance sphere

For parabola smoothing a parabola is inserted geometrically into the segment transition. This ensures a steady velocity transition within the tolerance radius.

The parabola is only inserted for straight line/straight line transitions.

5.2.14.3 Biquadratic smoothing

Bi-quad smoothing

Command	#set paramVertexSmoothing(<type>; <subtype>; <radius>)#
Parameter <type>	For biquadratic smoothing: 3
Parameter <subtype>	1: Constant tolerance radius [▶ 154] 2: Distance between intersection and vertex [▶ 154] 3: Adaptive tolerance radius [▶ 154]
Parameter <radius>	Max. radius of the tolerance sphere

With biquadratic smoothing there is no step change in acceleration in the axis components. With the same radius, a smaller input velocity may therefore be required than for parabolic smoothing.

The operating principle of the subtypes is identical to that of the parabolic subtypes.

5.2.14.4 Bezier curve of the 3rd order

Bezier curve of the 3th order

Command	#set paramVertexSmoothing(<type>; <subtype>; <radius>)#
Parameter <type>	for the Bezier curve of the 3th order: 4
Parameter <subtype>	1: Constant tolerance radius [▶ 154] 2: Distance between intersection and vertex [▶ 154] 3: Adaptive tolerance radius [▶ 154]
Parameter <radius>	Max. radius of the tolerance sphere

In case of the 3rd order Bezier curve a step change in acceleration appears in the axis components when the tolerance sphere is entered. The max. size is limited by the acceleration of the axis components and the C1 factor.

This blending can be used for all segment transitions. The subtypes 2 and 3 only work for straight line / straight line transitions.

● Acute angles at the segment transition

i The Bezier splines are generated by default, even at very acute angles. In order to avoid the dynamic values being exceeded, a considerable reduction velocity is required in this case. However, since the dynamics are held constant in the spline, the movement across the spline can be quite slow. In this case it is often practical to start the segment transition with an accurate stop. The command [AutoAccurateStop \[► 155\]](#) can be used to avoid having to calculate the angles manually.

5.2.14.5 Bezier curve of the 5th order

Bezier curve of the 5th order

Command	#set paramVertexSmoothing(<type>; <subtype>; <radius>)#
Parameter <type>	for the Bezier curve of the 5th order: 5
Parameter <subtype>	1: Constant tolerance radius [► 154] 2: Distance between intersection and vertex [► 154] 3: Adaptive tolerance radius [► 154]
Parameter <radius>	Max. radius of the tolerance sphere

With 5th order Bezier blending, **no** step change in acceleration occurs in the axis components on entry into the tolerance sphere. In other words, the path axis acceleration is always constant if blending is selected.

This blending can be used for all segment transitions. The subtypes 2 and 3 only work for straight line / straight line transitions.

● Acute angles at the segment transition

i The Bezier splines are generated by default, even at very acute angles. In order to avoid the dynamic values being exceeded, a considerable reduction velocity is required in this case. However, since the dynamics are held constant in the spline, the movement across the spline can be quite slow. In this case it is often practical to start the segment transition with an accurate stop. The command [AutoAccurateStop \[► 155\]](#) can be used to avoid having to calculate the angles manually.

5.2.14.6 Old Bezier blending type

● Functions for compatibility with existing projects

i These functions are provided for compatibility reasons. For new projects [Bezier curve of the 3rd order \[► 151\]](#) or [Bezier curve of the 5th order \[► 152\]](#) should be used.

Old Bezier blending with paramVertexSmoothing

Command	#set paramVertexSmoothing(<type>; <subtype>; <radius>)#
Parameter <type>	For Bezier Spline smoothing: 1
Parameter <subtype>	For Bezier Spline smoothing: 1
Parameter <radius>	radius of the tolerance sphere

Sample 1:

```
N10 R57=100
#set paramVertexSmoothing(1; 1;R57)#
```

Old Bezier blending with paramSplineSmoothing

With the aid of smoothing, it is possible to insert a Bezier spline automatically between two geometrical entries. It is only necessary to program the radius of the tolerance sphere. This describes the maximum permissible deviation from the programmed contour in the segment transition. The advantage of this type of smoothing as opposed to rounding with an arc is that there are no step changes in acceleration at the segment transitions.

The radius of the tolerance sphere can be altered at any time within the NC program, and can be switched off again by setting the radius to 0. If the radius is not reset to 0, it remains active until the next interpreter reset or TwinCAT restart.

Command	#set paramSplineSmoothing(<radius>)#
Parameter <radius>	Radius of the tolerance sphere

or alternatively

#set paramVertexSmoothing(...)

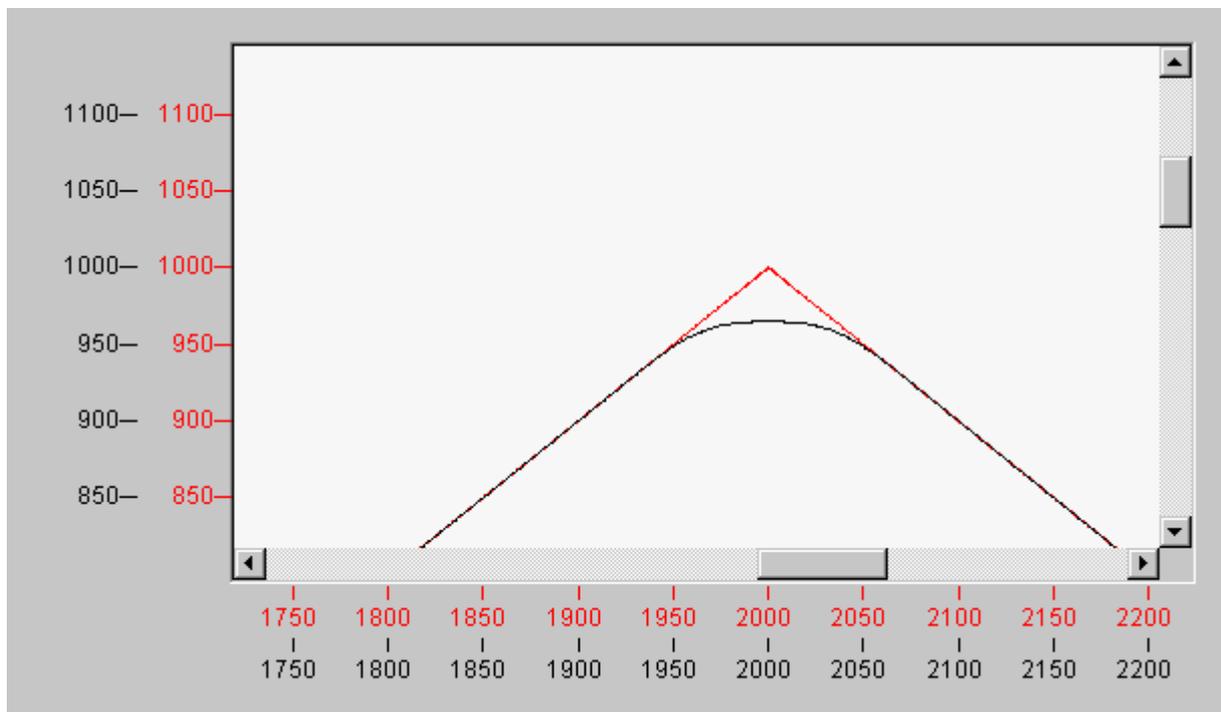
Sample 1:

```
N10 R57=100
#set paramSplineSmoothing(R57) #
```

Sample 2:

```
N10 G01 X0 Y0 F6000
N20 X1000
#set paramSplineSmoothing(100) #
N30 X2000 Y1000
N40 X3000 Y0
M30
```

The new parameter is valid from the transition between the subsequent two segments. In example 2, the new value for the tolerance sphere is applicable at the segment transition from N30 to N40. The diagram below shows a contour with and without spline at the segment transition.

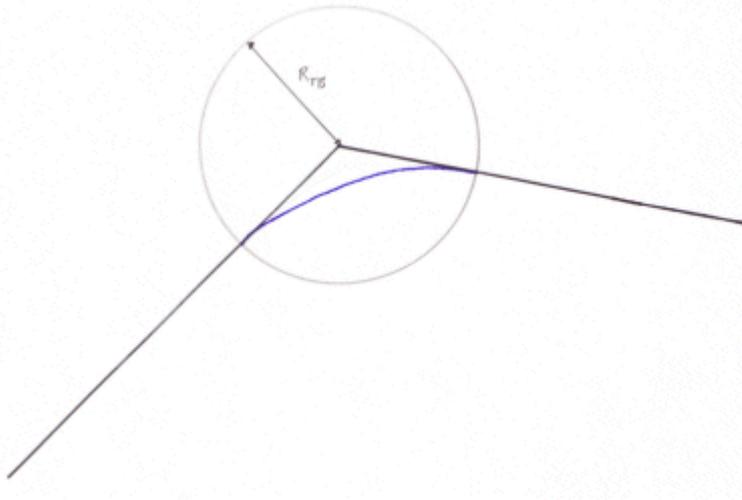


The splines are generated even at very sharp angles by default. In order to avoid the dynamic values being exceeded, a considerable reduction velocity is required in this case. However, as the dynamics are held constant, the movement across the spline can be quite slow. In this case it is often practical to start the segment transition with an accurate stop. In order to avoid manual calculation of the angles, an 'AutoAccurateStop [P_155]' command is available which can also be initiated via the NC program.

5.2.14.7 Subtypes

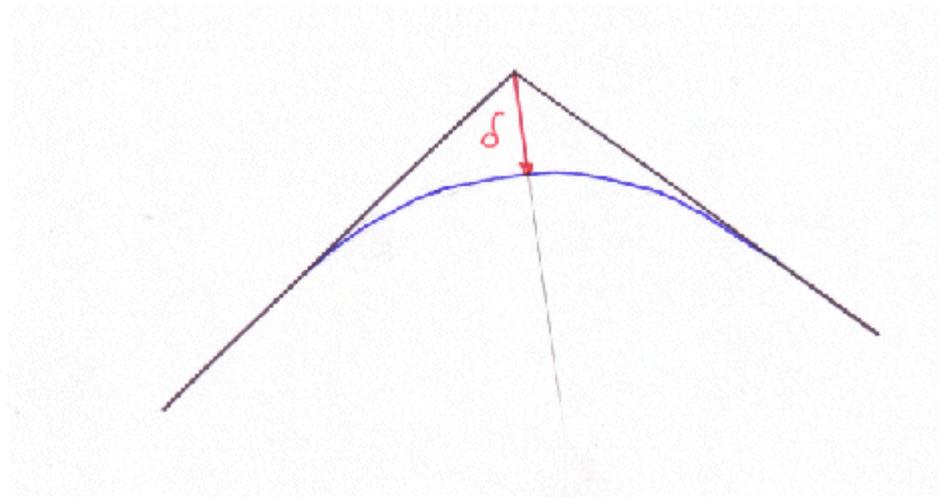
Constant tolerance radius (subtype 1)

If subtype 1 is selected, the maximum tolerance radius (R_{TB}) is used for blending. R_{TB} is reduced if and only if the input or output segment is less than $3 \cdot R_{TB}$.



Distance between intersection and vertex (subtype 2)

The distance between the programmed segment transition and the vertex of the parabola is specified with the subtype 2. The tolerance radius (R_{TB}) results from this. If a segment is too short, then the distance is shortened so that the tolerance radius is a max. of $1/3$.



Adaptive tolerance radius (subtype 3)

Within the tolerance radius (including constant tolerance radius) the system ensures that the maximum permissible acceleration is not exceeded. Depending on the deflection angle and the velocity, the maximum axis acceleration within the smoothing segment may be different. The aim of an adaptive tolerance radius is maximum acceleration during smoothing. In order to achieve this, the smoothing radius is reduced based on the programmed velocity and dynamics. In other words, if the programmed velocity is changed, the tolerance radius can also change. The override has no influence on the radius.

5.2.15 Circular Smoothing

It is possible with the aid of circular smoothing to insert an arc automatically between two straight lines. It is only necessary to program the radius of the arc.

The radius of the circular smoothing can be altered at any time within the NC program, and can be switched off again by setting the radius to 0. Rounding must be switched off before the end of the program or a decoder stop [► 166].

Command	#set paramCircularSmoothing(<radius>)#
Parameter <radius>	Radius of the circular smoothing arc

Sample:

```
N10 R57=4.5
#set paramCircularSmoothing(R57)#
...
#set paramCircularSmoothing(0)#
N1000 M02
```

Note When combined with cutter radius compensation, please note that first the radius compensation is calculated, then the circular smoothing is added. The smoothing radius thus refers to the TCP.

Note The old command paramGroupVertex continues to be supported. However, it cannot be used to transfer R parameters.

Syntax:

```
#set paramGroupVertex(<grp>,<radius>)#
```

The first parameter describes the group to which the circular smoothing refers. This value is currently always 1. The second parameter is used to specify the circular smoothing radius.

5.2.16 Automatic Accurate Stop

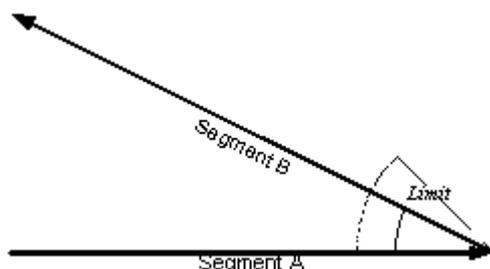
Command	#set paramAutoAccurateStop(<angle>)#
Parameter <angle>	Limit angle (in degrees) after which an accurate stop is inserted
Deselect	#set paramAutoAccurateStop(0)#

An accurate stop after a defined limit angle is inserted between two segments with the aid of the 'AutoAccurateStop' command.

For circle segments, the angle is calculated from the tangents at the points of entry and leaving.

Sample:

```
#set paramAutoAccurateStop(45)# (angle in
degrees)
N10 G01 X1000 Y0 Z0 F60000 (start position: X0 Y0 Z0)
N20 X0 Y500
...
```



An accurate stop is inserted between segments A and B in this example.

Application field:

This command should be used in conjunction with Bezier blending, if acute angles are programmed in the NC program.

See also:

- [Bezier curve of the 3th order \[► 151\]](#)
- [Bezier curve of the 5th order \[► 152\]](#)
- ['Old' Bezier curve \[► 152\]](#)

Note This function has not yet been implemented for segment transitions with a helix.

5.2.17 Delete Distance to Go

Command	DeIDTG
Cancellation	End of block

The DeIDTG (**delete distance to go**) command is activated block by block via the NC program. This command enables deleting of the residual distance of the current geometry from the PLC with the function block [\[tpDelDtgEx \[► 205\]](#). In other words, if the command is issued while the block is processed, the motion is stopped with the usual deceleration ramps. The NC program then processes the next block. An error message is generated if the PLC command is not issued during the execution of a block with "delete distance to go" selected.

The "delete distance to go" command always effects an implicit decoding stop, i.e. an exact positioning always occurs at the end of the block.

Sample:

```
N10 G01 X0 Y0 F6000
N20 DeIDTG G01 X2000
N30 G01 X0
```

Note DeIDTG must not be active when cutter radius compensation is active.

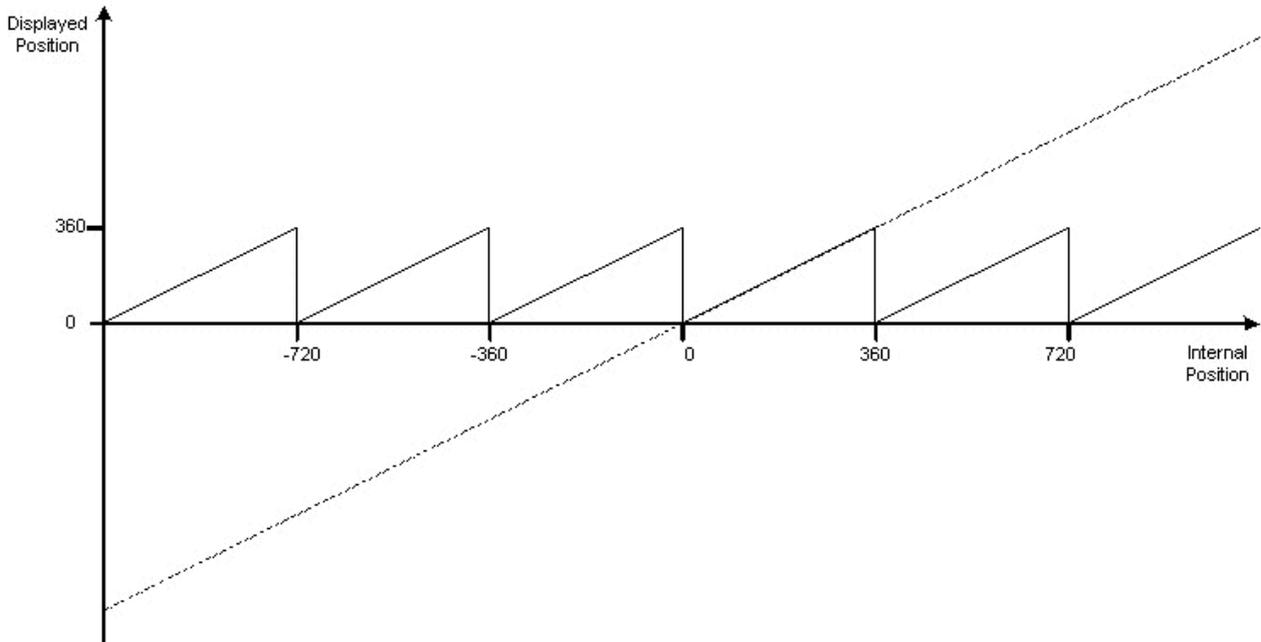
5.2.18 Modulo Movements

Command	MOD[<axis and target modulo position>]
Cancellation	End of block
Parameter 1	Axis for modulo operation
Parameter 2	Arithmetic sign for the direction of rotation (optional)
Parameter 3	Modulo position

The modulo position is programmed in the same way as normal positioning.

The MOD command is effective for specific blocks, and must be explicitly programmed for every axis that is selected for modulo operation. The modulo position's arithmetic sign specifies the direction of rotation.

- Positive sign: The axes moves in the 'larger' direction
- Negative sign: The axes moves in the 'smaller' direction
- Exception: The axis cannot move to modulo -0, since zero has no sign



Sample 1:

```
N10 G90
N20 G01 MOD[X200] Y30 F600
N30 G01 X200
```

N20 specifies a move in a positive direction for X to modulo position 200. Y moves to absolute position 30. In block N30, X is moved absolutely to position 200, i.e. **not** modulo.

● Modulo Operation Applicable to Q-Axes

i The Q-axes are the auxiliary axes. The MOD command can be applied to Q-axes. E.g.
 N20 G01 MOD[Q4=200] Y30 F600.

Modulo movements of more than 360 degrees

The MOD command also allows movements of more than a 360 degrees to be made.

Modulo position = number of necessary rotations * 360 + modulo position

Sample 2:

```
N10 G90
N20 G01 X3610 F6000
N30 R1=360
N40 G01 MOD[X=R1+20]
```

In this example, the X axis moves 370 degrees to modulo position 20.

Restrictions and notes of for modulo movements:

- No radius compensation may be active for the modulo axis.
- No zero shift may be active for the modulo axis.
- During relative programming (G91 [▶ 126]) the modulo command is not evaluated, so that the axis referred to in square brackets is treated as if the MOD command had not been given.

Modulo factor

The modulo factor is constant, and is 360.

5.2.19 Auxiliary axes

Auxiliary axes (also known as Q axes) can be added to an interpolation group in addition to the actual path axes (X, Y & Z). The auxiliary axis can be seen as a type of slave for the path, i.e. it has no direct influence on the path velocity. In addition to the 3 path axes, 5 auxiliary axes can also be interpolated for each channel.

The function block 'CfgBuildExt3DGroup [► 196]' from the library Tc2_NCI, for example, may be used for adding to the interpolation group from the PLC.

Syntax

The auxiliary axes are addressed as Q1..Q5 from the part program. The numerical value may be assigned directly, or an R-parameter.

Sample 1:

```
(start position X=Y=Z=Q1=0)
N10 G01 X100 Q1=47.11 F6000
...
```

If an NC block is programmed with one or more path axes and an auxiliary axis, both axes start **simultaneously** and also reach the destination **together**.

Swiveling of the auxiliary axes

The term "swiveling of the auxiliary axes" is used if the path length within a motion set is zero. This is often the case during 'swiveling' of a tool, with the feed angle relative to the contour being changed.

Since the path length is zero, there is no link to the path, and the movements of the auxiliary axes are calculated via a virtual path. However, this has no influence on the real path of X, Y and Z, but here too all auxiliary axes are started simultaneously and also arrive at the destination simultaneously.

Here too, the velocity is specified via the F-parameter and now refers to the auxiliary axis with the greatest travel distance.

Sample 1:

```
(start position X=Y=Z=Q1=Q2=0)
N10 G01 X100 F6000
N20 Q1=100 Q2=200 F3000
...
```

In N20, the velocity of Q2 is now 3000 and that of Q1 is 1500, since the travel distance is $Q1=Q2/2$.

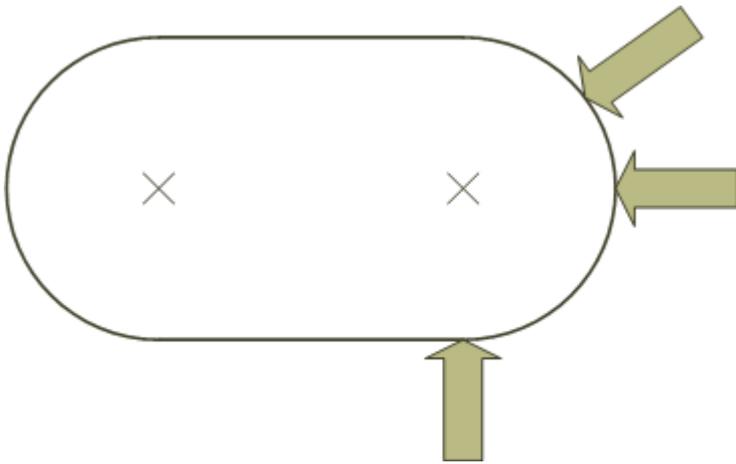
5.2.19.1 Calculation of the velocity

Initially, only the path axes (X, Y and Z) are considered for the calculation of the path velocity. The path and the travel distance of the individual auxiliary axes result in a fixed coupling ratio for each auxiliary axis within a segment. The target velocity of the auxiliary axis is thus also known. If this velocity is greater than the permitted maximum velocity for this auxiliary axis, the path velocity is reduced until the upper speed limit is adhered to. In other words, exceeding of the velocity limits of the auxiliary axes also has an indirect effect on the path velocity.

5.2.19.2 Path velocity at segment transitions

The reduction of the path velocity is explained below by means of an example. The contour of a stadium is particularly suitable for this purpose. The aim is for the feed angle of a tool relative to the path tangent to remain constant.

On the stadium straight, the orientation of the tool remains constant, i.e. the tool is not turned. In contrast, the orientation relative to the base coordinate system must be changed continuously within the circle. Assuming the path velocity in the transition between straight and circle is not reduced to zero, a step change in velocity is inevitably generated for the swiveling axis (but not for the path axes!).



This step change in velocity of the auxiliary axis is freely parameterizable and depends on the machine. Extreme cases would be for the path velocity at such segment transitions to be reduced to zero, or for the velocity not to be reduced at all.

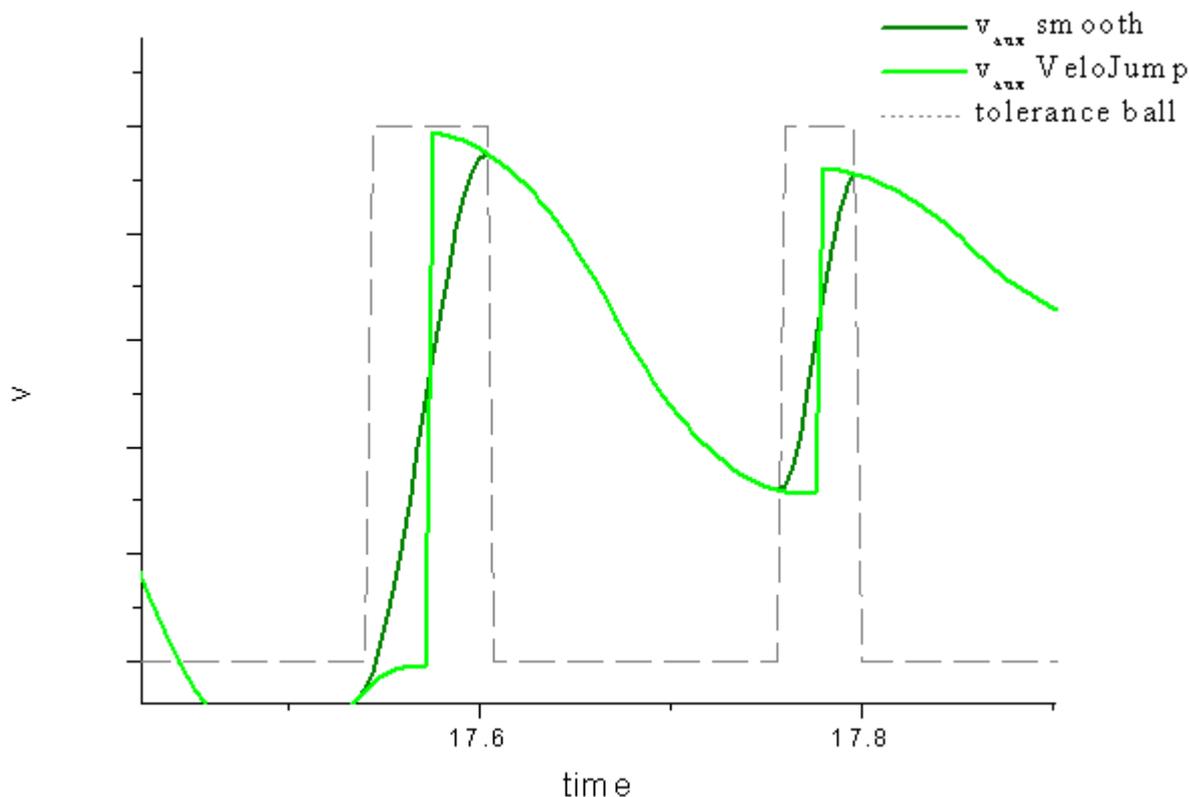
The global axis parameter 'VeloJumpFactor', which can be set individually for each axis, is used for the parameterization. The resulting velocity and the calculation is described in more detail in the TwinCAT NCI appendix on page [Parameterisation \[▶ 319\]](#).

Smoothing of the velocity at segment transitions

As has been described above, step changes in velocity can occur at the segment transitions. The size of these steps can be affected by the VeloJump parameter.

It is further possible for a tolerance sphere to be specified for every auxiliary axis. This sphere is symmetrical with the path at a segment transition. On entering this sphere, the velocity of the auxiliary axis is continuously modified to reach the set velocity at the exit of the sphere. The step changes in velocity are, in other words, eliminated. This does imply that the auxiliary axis is subject to a positional error when it is within the sphere. On entering the sphere the change to the new target velocity of the axis starts immediately. This avoids an overshoot in position, and the position is again precise at the borders of the sphere.

If it happens that the specified sphere is larger than 1/3 of the path, its radius is automatically restricted to that value.



Selection and Deselection

The tolerance sphere of the auxiliary axis is an axis parameter (IO: 0x108). It can be set in TwinCAT XAE axis interface Specification Axes.

● Parameterization of the axis parameters

i The parameters described here only take effect for axes that are in the interpolation group as auxiliary axes (Q1..Q5). For path axes (x,y,z) the parameters 'Velo Jump Factor', 'Tolerance ball auxiliary axis' and 'Max. position deviation, aux. axis' have no influence.

Diagnostics

It is possible to record the tolerance sphere of each auxiliary axis and the position error that results from this for diagnostic purposes. It is also possible to access the variables via ADS. They are to be found in the "Index offset" specification for Group state (Index group 0x3100 + ID) (IO: 0x54n and 0x56n).

Effect on VeloJump, if the size of the tolerance sphere is reduced

If the size of the tolerance sphere has to be reduced due to the given geometry, the VeloJump parameter is automatically adjusted for this segment transition. I.e. the path velocity in the transition is reduced more strongly. So the dynamics of the auxiliary axis is not exceeded for smaller tolerance spheres.

Positional deviation of the auxiliary axis if the tolerance sphere has to be reduced

The parameter 'maximum permitted positional deviation of the auxiliary axis' **only** takes effect if the tolerance sphere would have to be reduced due to the geometry.

The aim is to keep the path velocity high despite the smaller tolerance sphere, as long as the resulting position error does not exceed a threshold value. To this end the velocity of the auxiliary axis is kept constant and the position error is calculated. If the error is smaller than the maximum positional deviation the velocity is maintained for this segment transition, and the resulting position error is compensated in the next segment (the tolerance sphere then becomes unnecessary for this segment transition).

In the event that the position error would exceed the maximum deviation, the reduced tolerance sphere takes effect, including the VeloJump factor. And the path velocity is reduced if necessary.

Example 1:

Initial conditions:

- Set tolerance sphere: 5
- Max. positional deviation: 1
- The given geometry results in an effective tolerance sphere of 0.2, for example
- The potential positional deviation is 0.3

Resultant behavior:

- The path velocity remains at a constant high level
- The velocity of the auxiliary axis is kept constant
- For this transition no tolerance sphere is required
- The resulting positional deviation is compensated in the subsequent segment

Example2:

Initial conditions:

- Set tolerance sphere: 5
- Max. positional deviation: 1
- The given geometry results in an effective tolerance sphere of 1.2, for example
- The potential positional deviation is 1.1

Resultant behavior:

- The tolerance sphere is adjusted
- The VeloJump parameter is adjusted
- The path velocity is reduced at the segment transition
- There is **no** positional deviation that has to be compensated

Parameterization

The parameterization of the maximum permitted positional deviation is an Specification Axes. By default this feature is switched off (deviation = 0.0)

5.3 Supplementary Functions

5.3.1 M-Functions

Task: Signal exchange between NC and PLC

A range of equipment, such as collet chucks, drill drives, transport equipment etc. is best not driven directly by the NC, but indirectly, using the PLC as an adapting and linking controller. This makes it easy to consider feedback or safety conditions, without having to adapt the NC program, or even the NC system. The NC's M-functions involve digital signal exchange: functions are switched on or off, activated or deactivated. The transfer of numerical working parameters is not provided for here, but can be implemented in other ways (H-functions [[▶ 165](#)], T-numbers [[▶ 165](#)] etc.).

5.3.1.1 Available M-functions

Number of M-functions

A total of 160 M-functions are available per channel

M function	Meaning
0..159	Freely definable M-functions (except 2, 17, 30)

M function	Meaning
2	Program end
17	End of subprogram
30	Program end with deletion of all fast M functions

All M-functions (apart from the 3 pre-defined M-functions - M2, M17, M30) are freely definable. This means that, depending on the machine type, M8 can be used to switch on a cooling medium or indeed for any other functionality, for example. The machine manufacturer can select the function as required.

Like any other rules, the rules for reserved M-functions are read when TwinCAT is started. Additionally, an internal code is generated for these functions in the interpreter, which is responsible for the behavior described. These 3 M-functions therefore do not have to be described in the table. It makes sense to parameterize M2 and M30, even if M-functions are used.

● Priority of System Manager M-Functions



If M-functions are defined as well in the `m_defs.t<xx>` file as in the system manager, only those M-functions act that are defined in the system manager.

Types of M-functions

Basically, two signal exchange versions are available: fast signal bits, or transfer secured by handshake.

Secured Handshakes

M-functions that require feedback must be processed using bi-directional signal exchange between the NC and the PLC. If an M-function of type handshake is programmed, the velocity is reduced to 0 at this point. The PLC uses the [ltpIsHskMFunc \[▶ 225\]](#) function to check whether an M-function with handshake is present, in which case the number of the M-function is determined via [ltpGetHskMFunc \[▶ 218\]](#). The NC is in a waiting state and will not process further NC commands until the PLC has acknowledged the M-function. Processing of the NC program continues once acknowledgement has been received from the PLC ([ltpConfirmHsk \[▶ 204\]](#)).

This procedure permits the operation of the equipment controlled by the NC to be securely coordinated with the equipment controlled by the PLC. It is therefore advisable to acknowledge the M-function for starting the spindle (e.g. M3) once a minimum speed has been reached.

Since this kind of M-function involves synchronous functions, it is only ever possible for one M-function with handshake to be active in the NC program.

Fast signal bits

If no feedback is required from the PLC, fast signal bits can be used for activating M-functions. Since the NC does not have to wait for the PLC with these M-functions, [look-ahead \[▶ 123\]](#) can combine the segments. In this way it is possible to apply an M-function without velocity reduction.

A fast M-function can be detected in the PLC via `ltpIsFastMFunc`. This makes it possible to start any action from the PLC during a movement (laser on/off, cutter on/off, ...). Afterwards the M-function has to be reset with `ltpResetFastMFuncEx`. This makes it possible to use the M-function more than once.

A combination of fast signal bits and handshake is also possible. Since a handshake always requires acknowledgement from the PLC, the velocity has to be reduced to 0 in this case.

5.3.1.2 Resetting of M-functions

Resetting fast signal bits

The signal bits are active until they are reset explicitly, or until an M30 (end of program) or channel reset is executed.

Resetting with reset list

Each M-function can reset up to 10 fast M-functions. If cooling medium is switched on with M8, for example, the cooling medium can be switched off again with M9. To this end simply enter M8 in the reset list for M9.

i Resetting a Fast M-Function Requires a Fast M-Function

A fast M-function can be reset only by a fast M-function. There is no alternative possibility in this case to reset a fast M-function by a non-fast handshake M-function.

Automatic reset

During parameterization of the M-function an 'auto-reset flag' can be set. This means that the M-function is automatically reset at the end of the block.

In order for the PLC to be able to see the signal, the duration of the motion block must be long enough, or this M-function is combined with a handshake. The handshake may come from the same or a different M-function.

Reset from the PLC

The fast M-functions can be reset from the PLC via the [ltpResetFastMFunc \[► 281\]](#) function block. For reasons of transparency, mixed resets using via PLC and NC should be avoided.

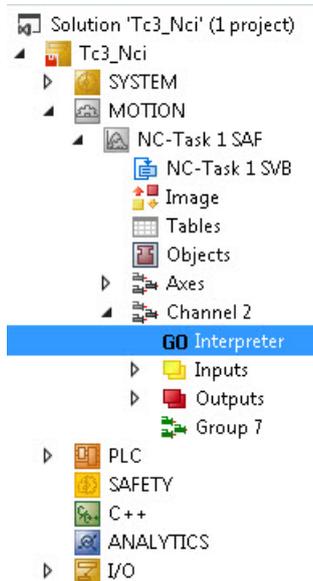
Delete all pending M-functions

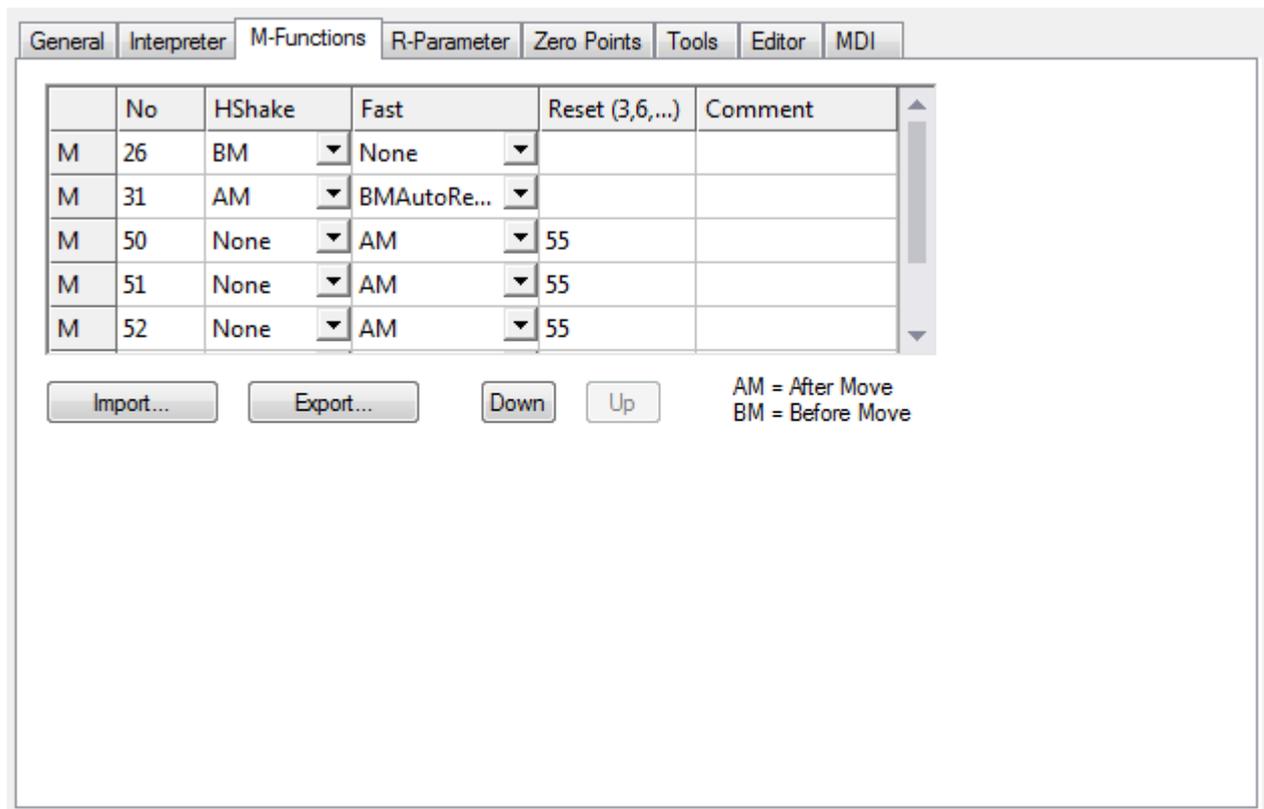
A channel stop and a channel reset are used to reset all pending M-functions. This is true for the 'handshake' type M-functions, and also for the fast signal bits. If the NC program is terminated properly with M30, all M-functions are also cleared.

5.3.1.3 Parameterization of M-functions

The M-functions are parameterized in TwinCAT XAE. A dedicated M-function table is used for each interpolation channel.

Activation of the TwinCAT configuration is required to enable a configuration of M-functions.





No

Number of M-function to be parameterized. The value must be between 0 and 159

HShake

If a value other than 'None' is entered, the M-function is of type 'Handshake'

- *None*: No handshake
- *BM* (Before Move): If a movement is programmed in the same block, the handshake is completed **before** the movement
- *AM* (After Move): If a movement is programmed in the same block, the handshake is completed **after** the movement

Fast

If a value other than 'None' is entered, a 'fast signal bit' type M-function is executed

- *None*: No fast M-function is executed
- *BM* (Before Move): If a movement is programmed in the same block, the output is completed **before** the movement.
- *AM* (After Move): If a movement is programmed in the same block, the output is completed **after** the movement.
- *BMAutoReset* (Before Move & Automatic Reset): If a movement is programmed in the same block, the output is completed **before** the movement. In addition, the M-function is automatically canceled at the end of the block, i.e. the M-function is active on a per-block basis. In order to ensure that the PLC recognizes the M-function, the duration of the associated motion block must be long enough (at least 2 PLC cycles), or an additional M-function with handshake should be programmed.
- *AMAutoReset* (After Move & Automatic Reset): This parameterization is only meaningful if either an M-function of type handshake is programmed at the same time (or parameterized), or if the M-function is only used for resetting other M-functions. Without an additional handshake the PLC will usually not be able to detect this M-function.
- All other combinations can be selected for compatibility reasons.

Reset

Up to 10 M-functions can be entered for cancellation when a reset is called.

Note In the event that no reset-signal-bit is in fact set, the bits to be cleared are reset immediately before setting the new signal bits.

Import/Export

The M-functions are parameterized individually for each channel. The parameterization can be transferred to other channels via the import/export function.

5.3.1.4 Combination of M functions

- Within each line, only **one** 'handshake' type M function must be programmed!
- Within a single line, up to 10 'signal bit' M functions may be programmed
- A combination of the two options above is allowed

Sample:

```
N10 G01 X1000 F60000
N20 M10 M11 M12 X2000 (M10 & M11 are signal bits)
(M12 is of type handshake)
M30
```

Examples of meaningful and practically applicable rule combinations:

- An M-function is to be active for the duration of a movement and then be automatically cleared. Select 'None' in the HShake column and 'BMAutoReset' in the Fast column. The signal bit generated could, for instance, control a glue application valve.
- An M-function starts a drill motor, and the subsequent movements may only be started after an appropriate run-up time, and then only when the drill is ready for operation. Select 'BM' in the HShake column. The PLC acknowledges the request after a certain delay time and only if the frequency converter is ready for operation.
- A drill motor is started with an M-function. In order not to have to wait for the drive to run up, the M-function is programmed in the block before the one for the drill movement. In the following movement (the drill movement itself) it is however still essential to ensure that the drive has reached its full rotation speed. For this variant either two different M-functions have to be used (lead signal as signal bit, safety query as handshake) or a Fast 'BMAutoReset' and HShake 'AM' M-function is used.

5.3.1.5 Behavior in case of an error

If a runtime error occurs during the execution of an NC program (e.g. following error monitoring is activated), the NC program is interrupted. In this case the M functions, provided they are set, remain pending. This means that the PLC program may have to ensure that M functions are not executed.

5.3.2 H, T and S Parameters

H-, T- and S-parameters are used to transfer parameters from the NC interpreter to the PLC.

In this context the H-parameter stands for auxiliary parameter and is of type DINT (32 bit signed).

The T and S parameters are of type WORD, and stand for Tool and Spindle.

Sample:

```
H=4711
R1=23
S=R1
T4711
```

Note No R-parameter can be assigned for the T-parameter. Furthermore, the assignment is made without assignment operator ('=').

T- and S-parameters take effect at the start of a block, H-parameters take effect at the end of the programmed block.

5.3.3 Decoder stop

Code	Function
@714 [▶ 166]	Decoder stop
@716 [▶ 166]	Decoder stop with axis position rescan
@717 [▶ 167]	Decoder stop with trigger event, conditional decoder stop

5.3.3.1 Decoder stop (@714)

The interpreter offers the option to execute a decoder stop in the NC program. In this case the interpreter waits until a certain external event occurs. Execution of the NC program does not continue until this event has taken place.

A decoder stop can be used, for instance, to switch [block skipping \[▶ 123\]](#) on or off from the PLC, or to re-assign [R parameters \[▶ 130\]](#).

Two events are available for continuing processing:

- Acknowledgement of an [M-function \[▶ 161\]](#)
- SAF task is empty

Acknowledgement of an M-function

Decoding of the NC program is interrupted until the [M-function \[▶ 161\]](#), which is programmed immediately prior to the decoder stop, is acknowledged. In other words, the M-function must be of type "handshake".

Sample 1:

```
N10...
N20 M43 (M-function with handshake)
N30 @714 (decoder stop)
N40 ...
```

SAF task is empty

The decoder stop does not necessarily have to be programmed in conjunction with an M-function. If the SAF task runs out of travel commands, an event is sent to the interpreter. This event causes the interpreter to start up again.

Note The decoder stop must not be programmed when the tool compensation or circle smoothing are active, because they wouldn't work anymore.

5.3.3.2 Decoder Stop with Axis Position Rescan (@716)

In addition to the common decoder stop (see [Decoder stop \(@714\) \[▶ 166\]](#)), there is a decoder stop at which the axis positions of the interpolation channel are read again. This stop is required, if, for example, axes are moved during a tool change via PTP and are subsequently not returned to the old position. Another possible application is a change in axis configuration via an M function (with handshake).

If a decoder stop with rescan is programmed, it is essential to program an M-function with handshake immediately before it.

Sample 2:

```
N10...
N20 M43 (M function with handshake carries out a tool change, for
example)
N30 @716 (Decoder stop with rescan)
N40 ...
```

Note The decoder stop must not be programmed when the tool compensation or circle smoothing are active, because they wouldn't work anymore.

5.3.3.3 Decoder Stop with external trigger event (@717)

Sometimes the question of whether the NC part of the program must wait or can continue may depend, for instance, on events in the PLC. With the two types of [M-functions](#) [▶ 161](#) this can give rise to the following problems:

- Handshake: Because of the M-function's handshake the path velocity must be brought to zero at the location where the M-function is programmed, after which confirmation is awaited from the PLC.
- On The Fly (also known as a fast M-function): Because no confirmation from the PLC is waited for, there is also no way for the partial program to wait for the PLC.
- Even a combination of the two types of M-function does not help here.

Sample:

During positioning with a flying M-function, a process A is initiated by the NC partial program. It is assumed here that the set of processes in the NC program is typically long enough for process A to be completed in the PLC. If A is ready, then the NC partial program should execute the next segment with look-ahead. In case A is not ready, however, then the NC should stop at the end of the segment and wait until process A has finished. It is exactly this scenario that can be implemented with the command @717. The PLC here sends the so-called 'GoAhead [▶ 222](#)' command when process A has finished.

```
N10 ...
N20 G0 X0 Y0 Z0
N30 G01 X500 F6000
N40 M70 (flying M-function that triggers process A)
N50 G01 X700
N60 @717 (decoder stop with external trigger event)
N70 G01 X1000
N80 ...
```

If the GoAhead signal reaches the PLC early enough, then blocks N50 and N70 are linked by look-ahead, and the path velocity is not then reduced. If the signal arrives during the deceleration phase of N50, then the velocity is once more increased. Otherwise, the machine waits for the signal from the PLC.

Note The decoder stop must not be programmed when the tool compensation or circle smoothing are active, because they wouldn't work anymore.

The function block 'ltpGoAheadEx' returns the error code 0x410A, if no @717 is present in the interpreter at the time of the call.

5.3.4 Jumps

Code	Function
@100 ▶ 167	Unconditional jump
@121 ▶ 168	Jump if unequal
@122 ▶ 168	Jump if equal
@123 ▶ 168	Jump if less or equal
@124 ▶ 168	Jump if less
@125 ▶ 168	Jump if greater or equal
@126 ▶ 168	Jump if greater
@111 ▶ 168	Case block

Unconditional jump

Command	@100
Parameter	K or R

The parameter describes the jump destination. This must have an indication of direction ('+' or '-').

Sample 1:

```
N10 ..
...
N120 @100 K-10
```

In this example, execution continues from line 10 after line 110 has been interpreted. The sign indicates the direction in which the line to be searched can be found.

Jump if unequal

Command	@121	
Parameter 1	R<n>	Comparison value
Parameter 2	K or R<m>	Comparison value
Parameter 3	K	Jump destination with direction indication

Sample 2:

```
N10 ..
...
R1=14
N120 @121 R1 K9 K-10
N130 ...
```

Jump if equal

cf. [Jump if not equal \[► 168\]](#)

Jump if less or equal

cf. [Jump if not equal \[► 168\]](#)

Jump if less

cf. [Jump if not equal \[► 168\]](#)

Jump if greater or equal

cf. [Jump if not equal \[► 168\]](#)

Jump if greater

cf. [Jump if not equal \[► 168\]](#)

Case block

Command	@111	
Parameter 1	R<n>	Comparison value
Parameter 2	K or R<m>	First comparison value
Parameter 3	K	First jump destination
Parameter 4	K or R<m>	Second comparison value
...		

Sample 3:

```
N100 R2=12 (R2=13) (R2=14)
N200 @111 R2 K12 K300
K13 K400
K14 K500

N300 R0=300
N310 @100 K5000

N400 R0=400
N410 @100 K5000
```

```
N500 R0=500
N510 @100 K5000

N5000 M30
```

A case block is made in line 200. If R2 = 12 a jump is made to line 300.

If R2 = 13, the jump destination is line 400. If R2 = 14, the jump destination is line 500.

In the event that none of the conditions is satisfied, execution simply continues with the next line (in this case, line 300).

5.3.5 Loops

The various types of loop are described below.

Code	Loop type	Aborting condition
@131	While Loop [▶ 169]	while equal
@132	While Loop [▶ 169]	while not equal
@133	While Loop [▶ 169]	while greater
@134	While Loop [▶ 169]	while greater or equal
@135	While Loop [▶ 169]	while less
@136	While Loop [▶ 169]	while less or equal
@141	Repeat Loop [▶ 170]	repeat until equal
@142	Repeat Loop [▶ 170]	repeat until not equal
@143	Repeat Loop [▶ 170]	repeat until greater
@144	Repeat Loop [▶ 170]	repeat until greater or equal
@145	Repeat Loop [▶ 170]	repeat until less
@146	Repeat Loop [▶ 170]	repeat until less or equal
@151	For-To Loop [▶ 170]	
@161	For-DownTo Loop [▶ 170]	

Loops can be nested.

While loops

Command	@13<n>	where 1 <= n <= 6
Parameter 1	R<m>	Comparison value
Parameter 2	K or R<k>	Comparison value
Parameter 3	K	Jump destination for the case that the condition is not met

A while loop is executed for as long as the condition is satisfied. The test is made at the beginning of the loop. If the condition is not or no longer met, a jump to the specified line takes place (parameter 3).

At the end of the While loop an unconditional jump ([@100 \[▶ 167\]](#)) must be programmed. The target of this jump is the line number of the while loop.

The loop's exit condition is specified with <n>.

Sample 1:

```
N100 R6=4
N200 @131 R6 K4 K600 (K600 is the target of the jump, when the condition is no longer satisfied)
N210 ...
N220 @100 K-200
```

```
N600 ...
```

```
N5000 M30
```

The loop (lines 200 to 220) is repeated for as long as $R6 = 4$. Once the condition is no longer satisfied, execution jumps to line 600.

Repeat loops

Command	@14<n>	where $1 \leq n \leq 6$
Parameter 1	R<m>	Comparison value
Parameter 2	K or R<k>	Comparison value
Parameter 3	K	Jump destination at the start of the loop

In a repeat loop, the interrogation takes place at the end of the loop. This means that the loop is executed at least once. The loop is only ended, to continue with the rest of the program, when the condition is satisfied.

Sample 2:

```
N200 ...
```

```
N210 ...
```

```
N300 @141 R6 K25 K200
```

The loop is repeated until $R6 = 25$. The second constant in line 300 gives the jump target (the start of the loop).

For-To loops

Command	@151 <variable> <value> <constant>
---------	------------------------------------

A for-to loop is a counting loop that is executed until the *variable* equals the *value*. The test is made at the beginning of the loop. If that condition is satisfied, execution jumps to the line specified by the *constant*.

The variable must be incremented (@620) at the end of the loop, and there must be an unconditional jump to the start of the loop.

Sample 3:

```
N190 R6=0
```

```
N200 @151 R6 K20 K400
```

```
N210 ...
```

```
N290 @620 R6 (increment R6)
```

```
N300 @100 K-200
```

For-Downto Loops

Command	@161 <variable> <value> <constant>
---------	------------------------------------

A for-downto loop is a counting loop. The behaviour is similar to that of a for-to loop. The difference is merely that the variable is decremented (@621) by 1 at the end of the loop.

5.3.6 Subroutine techniques

As in other fields, it is also valuable in NC programming to organize frequently used command sequences as subroutines. This makes it possible to employ pre-prepared and tested functions in various workpiece programs.

Subroutines are identified within a program by a number. This number must be unique: there must be only one subroutine with a particular number (1..>2.000.000.000).

As interpretation proceeds, the calling program is suspended. The text of the subroutine is worked through, as often as necessary. Processing then returns to the calling program after the call location.

It is of course possible for one subroutine to call another subroutine. This call is executed in a similar way. This causes a stack of return information to be created. For technical reasons this nesting of subroutines is presently limited to 20 levels.

Definition of a Subroutine

The code for a subroutine can be written to the same file as the calling program. In this case the subroutine is linked directly: it is automatically also loaded as the file is read. If it is to be made generally available then it must be written in its own file that must be located in the CNC directory.

The name of the file begins with the letter 'L', and continues with a string of digits. This digit string must repeat the subroutine number, without any leading '0's.

The code should contain a label to indicate the starting point of the subroutine. Like the file name, it consists of the letter 'L' and the digit sequence described above.

The interpreter starts immediately after this label.

Subroutine syntax:

```
(Date: L2000.NC)
L2000
N100...
N110...
...
N5000 M17 (return command)
```

Calling a Subroutine

The following syntax must be used to call a subroutine from some block within the NC program. It is important that the expression "L2000" does not stand at the start of the line, in order to avoid confusion with a subroutine label.

```
(syntax of the subroutine call)
N100 L2000
```

In the following sample the expression "P5" causes the subroutine to be repeated 5 times.

```
(n-fold subroutine call (here: 5- fold))
N100 L2000 P5
```

Dynamic subroutine call

In some cases the subroutine to be called is not known until runtime. In this case the subroutine can be called with an R-parameter, thereby avoiding the need for a CASE instruction. The value for R must be allocated or calculated in a dedicated line.

```
(Dynamic call of a subroutine)
N099 R47=R45+1
N100 L=R47
```

Parameter passing

Parameters are passed to subroutines with the aid of [R-parameters \[► 130\]](#). Note that R-parameters are not saved automatically (see [Rescuing R-parameters \[► 130\]](#)).

Use of Parameters

R-parameters can, in general, be freely used within subroutines. This has a number of consequences that can lead to errors if they are not borne in mind. On the other hand their careful use offers the NC-programmer a range of useful working techniques.

Results of Subroutines

If an R-parameter is changed without its contents being saved and restored, the change is effective after a return from the subroutine. If this was not intended, the result can be machine behavior that was not planned.

This feature can however be deliberately used in order to make the continuation of the processing dependent on the results of a subroutine. No restriction need be considered here other than those on the R-parameters.

Sample:

```
N100 L2000
N110 R2=R3+R4
...
N999 M30

L2000
N10 R3=17.5
N20 R4=1
N99 M17
```

Values are specified here in a subroutine. The values are then used in the calling program.

Ending a Subroutine

A subroutine is ended with M17.

5.3.7 Dynamic Override

Command	DynOvr=<value> or DynOvr = R<n>
Cancellation	DynOvr=1

Sample:

```
N10 G01 X100 Y200 F6000
N20 DynOvr=0.4
N30 G01 X500
```

'DynOvr' can be used to make percentage changes to the dynamic parameters of the axes in the group while the NC program is running. This also results in new values for the motion dynamics. The new dynamic values become valid, without any stop, when the line is executed. This means, for the example illustrated above, that in block 10 the old values will still be used for the deceleration, while the new values will be used for acceleration in block 20.

Scope of Definition

$0 < \text{DynOvr} \leq 1$

See also [change in path dynamics \[► 172\]](#).

5.3.8 Altering the Motion Dynamics

Command	#set paramPathDynamics
Parameter <acc>	Value of the maximum permitted path acceleration in mm/s ² .
Parameter <dec>	Value of the maximum permitted deceleration in mm/s ² .
Parameter <jerk>	Value of the maximum permitted jerk in mm/s ³ .

Sample:

```
N10 G01 X100 Y200 F6000
N15 R4=3000
N20 #set paramPathDynamics( 700; 700; R4 )#
N30 G01 X500
```

The 'paramPathDynamics' command can be used to alter the motion dynamics as the NC program is running. The new dynamic values become effective as from the line in which they are programmed. For the example illustrated, this means that the whole of block 10 is still treated with the default values. The new parameters are used for block 30 from the start of the segment.

This command limits all path axes to the parameterized dynamic values, although the path itself can have higher dynamics, depending on its orientation. The dynamics of auxiliary axes remains unchanged.

See also [dynamic override](#) [▶ 172].

Note The dynamic values changed via the NC program remain active until the interpreter is next reset and/or TwinCAT has been restarted.

Note The old command 'paramGroupDynamics' continues to be valid. However, it cannot be used to transfer R parameters.

Command	#set paramGroupDynamics(<grp>,<acc>,<dec>,<jerk>)#
Parameter <grp>	Group for which the alteration of the motion dynamics is to be effective. Presently always 1.
Parameter <acc>	Value of the maximum permitted path acceleration in mm/s ² .
Parameter <dec>	Value of the maximum permitted deceleration in mm/s ² .
Parameter <jerk>	Value of the maximum permitted jerk in mm/s ³ .

Sample:

```
N10 G01 X100 Y200 F6000
N20 #set paramGroupDynamics( 1, 700, 700, 3000 )#
N30 G01 X500
```

Change in axis dynamics

Command	#set paramAxisDynamics
Parameter <axis>	Axis in the interpolation group: X: 0 Y: 1 Z: 2 Q1: 3 ... Q5: 7
Parameter <acc>	Value of the maximum permitted acceleration in mm/s ²
Parameter <dec>	Value of the maximum permitted deceleration in mm/s ² .
Parameter <jerk>	Value of the maximum permitted jerk in mm/s ³ .

Sample:

```
N10 G01 X100 Y200 F6000
N15 R4=30000
N20 #set paramAxisDynamics( 0; 1500; 1400; R4 )#
N30 G01 X500
```

'paramAxisDynamics' can be used to change the axis dynamics at runtime. Generally the behavior is the same as for 'paramPathDynamics', except that here the dynamics can be specified individually for each axis.

5.3.9 Change of the Reduction Parameters

C0 reduction [▶ 174]
C1 reduction [▶ 174]

C2 reduction [► 175]

C0 reduction

In some types of machine it is not absolutely necessary to reduce the path velocity to 0 at knee-points. 2 reduction methods are available

- VeloJump
- DeviationAngle

VeloJump

Command	#set paramVeloJump(<C0X>; <C0Y>; <C0Z>)#
Parameter <C0X>	Reduction factor for C0 transitions: X axis: $C0X \geq 0.0$
Parameter <C0Y>	Reduction factor for C0 transitions: Y axis: $C0Y \geq 0.0$
Parameter <C0Z>	Reduction factor for C0 transitions: Z axis: $C0Z \geq 0.0$

The 'paramVeloJump' command can be used to alter the velocity step change factors as the NC program is running. The new values come into effect via the block execution in the programmed line. You can find further details of the means of operation in the appendix under [Parameterization \[► 319\]](#).

Sample:

```
N10 G01 X100 Y200 F6000
N20 R2=4.5
N30 #set paramVeloJump( 1.45; R2; R2 )#
N40 G01 X500
```



Resetting parameters

The VeloJump parameters changed via the NC program remain active until the interpreter is next reset and/or TwinCAT has been restarted.

DeviationAngle (not yet released)

Command	#set paramDevAngle(<C0Factor>; <AngleLow>; <AngleHeigh>)#
Parameter <C0Factor>	Path reduction factor for C0 transitions: $1.0 \geq C0 \geq 0.0$
Parameter <AngleLow>	Angle in degrees from which reduction takes effect: $0 \leq \varphi_l < \varphi_h \leq \pi$
Parameter <AngleHeigh>	Angle in degrees from which reduction to $v_{link} = 0.0$ takes effect: $0 \leq \varphi_l < \varphi_h \leq \pi$

The 'paramDevAngle' command is used to describe the parameters for the C0 reduction. In contrast to the VeloJump reduction method, in which the velocity step change is influenced directly, in the DeviationAngle method the velocity step change depends upon the angle. You can find further details of the means of operation in the appendix under [Parameterization \[► 319\]](#).

Sample:

```
N10 G01 X100 Y200 F6000
N20 #set paramDevAngle(0.15; 5; 160 )#
N30 G01 X500
```



Resetting parameters

The DeviationAngle parameters changed via the NC program remain active until the interpreter is next reset and/or TwinCAT has been restarted.

C1 reduction factor

Command	#set paramC1ReductionFactor(<C1Factor>)#
Parameter <C1Factor>	C1 reduction factor

The 'paramC1ReductionFactor' command is used to change the C1 reduction factor while the NC program is running.

The new parameter comes into effect at the segment transition at which the reduction factor is programmed. In the example shown, the new value for the C1 reduction is therefore already effective in the segment transition from N10 to N30.

A floating point value or an 'R parameter' can be provided as parameter.

You can find further details of the means of operation in the appendix under [Parameterization \[► 319\]](#).

Sample:

```
N10 G01 X100 Y200 F6000
N20 #set paramC1ReductionFactor( 0.45 )#
N30 G01 X500
```

● Resetting parameters

i The C1 reduction factor changed via the NC program remains active until the interpreter is next reset and/or TwinCAT has been restarted.

C2 reduction factor

Command	#set paramC2ReductionFactor(<C2Factor>)#
Parameter <C2Factor>	C2 reduction factor

The 'paramC2ReductionFactor' command is used to change the C2 reduction factor while the NC program is running.

The command takes effect in the segment transition for which the reduction factor is programmed. In the example shown, the new value for the C2 reduction is therefore already effective in the segment transition from N10 to N30.

A floating point value or an 'R parameter' can be provided as parameter.

Sample:

```
N10 G01 X100 Y200 F6000
N20 #set paramC2ReductionFactor( 1.45 )#
N30 G01 X500
```

● Resetting parameters

i The C2 reduction factor changed via the NC program remains active until the interpreter is next reset and/or TwinCAT has been restarted.

5.3.10 Change of the Minimum Velocity

Command	#set paramVeloMin(<VeloMin>)#
Parameter <VeloMin>	Minimum path velocity

The 'paramVeloMin' command can be used to alter the minimum path velocity while the NC program is running. The new velocity comes into effect via the block execution in the programmed line.

A floating point value or an 'R parameter' can be provided as parameter.

Sample:

```
N10 G01 X100 Y200 F6000
N20 #set paramVeloMin( 2.45 )#
N30 G01 X500
```

● Resetting parameters

i The minimum velocity changed via the NC program remains active until the interpreter is next reset and/or TwinCAT has been restarted.

i Programming the velocity

The unit of velocity is mm/sec and is therefore equivalent to the usual XAE units.

5.3.11 Read Actual Axis Value

Command	@361	
Parameter 1	R<n>	R parameter to which the actual axis value is assigned
Parameter 2	K<m>	Constant for the axis coordinate that is to be read 0: X axis 1: Y axis 2: Z axis 3: Q1 axis 4: Q2 axis ... 7: Q5 axis

Sample 1:

```
N10 G0 X0 Y0 Z0 F24000
N30 G01 X1000
N40 @361 R1 K0 (read position of x axis)
N50 R0=X
N60 G01 X=R0+R1
N70 M30
```

A decoder stop is implicitly executed by @361 command. This ensures that, in this example, the position is read when block N30 has been processed.

A possible application would be in combination with the deletion of any remaining travel.

Read actual axes value without decoder stop

Command	#get PathAxesPos(R<a>; R; R<c>)#	
Parameter 1	R<a>	R parameter to which the actual axes value of the X axis is assigned
Parameter 2	R	R parameter to which the actual axes value of the Y axis is assigned
Parameter 3	R<c>	R parameter to which the actual axes value of the Z axis is assigned

The command #get PathAxesPos()# reads the current actual positions of the path axes (X, Y & Z). It behaves similarly to @361, with the difference that this command does not trigger an implicit decoder stop. This means that the programmer must himself ensure that at the time when the command is being processed in the interpreter the axes have not yet moved, or else a decoder stop (@714) must be programmed in the block before this command.

#get PathAxesPos()# is an alternative to @361, but it is linked to certain specific conditions.

Sample 2:

```
@714(optional)
N27 #get PathAxesPos( R0; R1; R20 )#
```

Note If a path axis is not assigned (e.g. no axis is assigned to Z) the value 0 is passed to the associated R parameter.

5.3.12 Skip virtual movements

Command	#skip VirtualMovements(<parameter>)#
Parameter	0 (default): virtual movements are "completed". 1: Virtual movements are skipped

Movements of unavailable but programmed main axes (X, Y & Z) can be skipped with the command 'skip VirtualMovements'.

Sample:

The interpolation group (CfgBuildGroup) contains only assignments for the X and Y axis. The Z axis is **not** assigned, but programmed in the parts program.

```
(Startposition X0 Y0 Z0)
N10 #skip VirtualMovements(1)#
N20 G01 X100 Y200 F6000
N30 G01 Z1000 (virtual movement, because z is not assigned)
N40 G01 X500
```

Segment N30 is skipped during execution of this program.

5.3.13 Messages from NC program

Command	#MSG (<message level>; <mask>; "<text>")#
<message level>	<ul style="list-style-type: none"> ITP The message is issued from the interpreter. This means that the message generally appears well before the execution in the NC kernel. NCK The message is issued from the NC kernel when the NC block is executed. This means it appears at the same time as the block execution (SAF)
<mask>	STRING
<text>	the text to be displayed

```
N10 G0 X0 Y0
N20 G01 X100 Y0 F6000
N30 #MSG( NCK; STRING; "this is a text")#
N40 G01 X200 Y-100
```

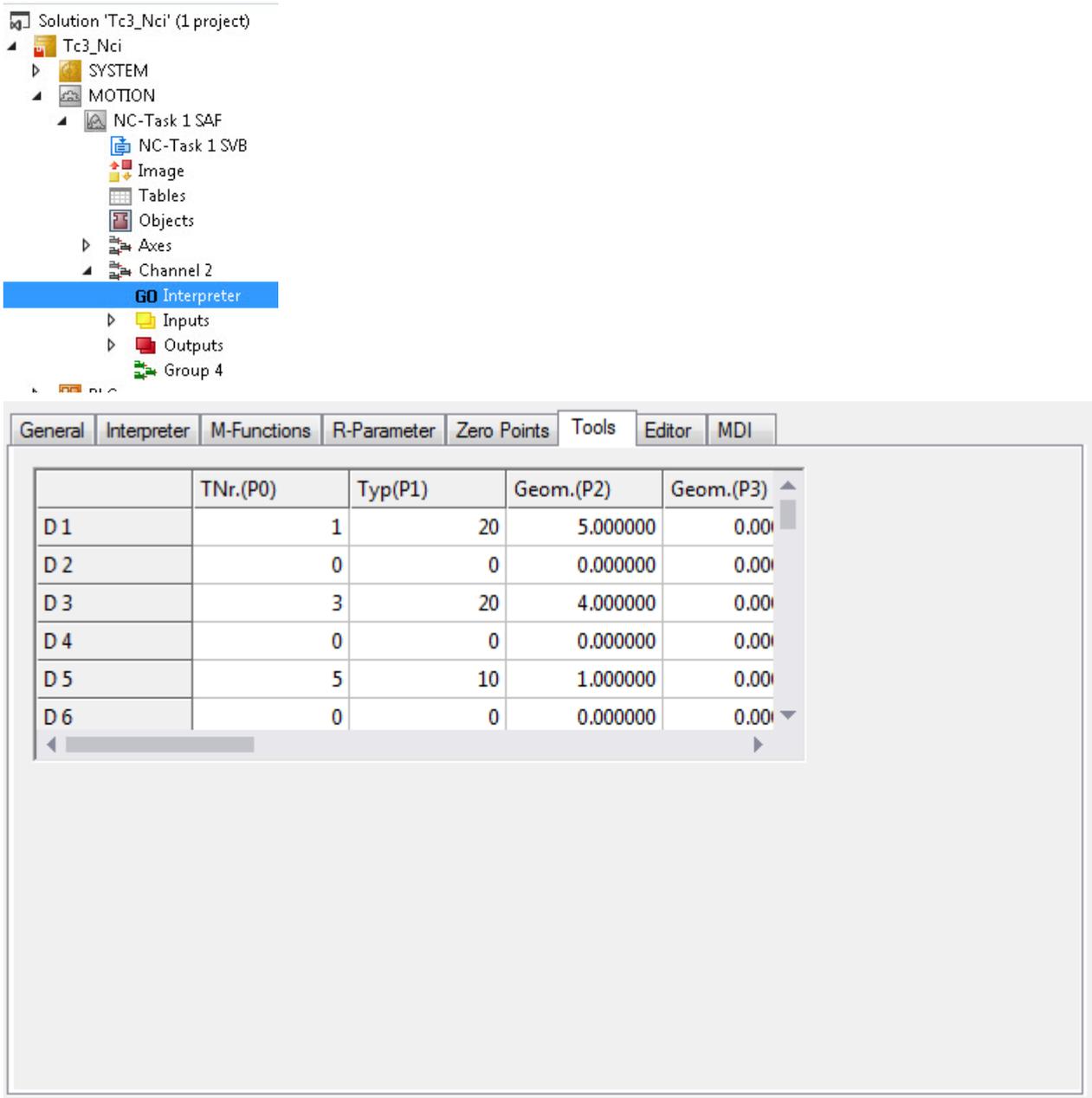
The text can **not** be used to transfer further parameters (e.g. R-parameters).

Internally the message is handled like a note.

5.4 Tool Compensation

5.4.1 Tool Data

The NC has 255 memory locations (D1..D255) available for each channel for tool data. The parameters for the tool data can be written directly in the XAE. The data is saved as an ASCII file (<channel ID>.wz) which is kept in the TwinCAT\CNC directory. These files are automatically loaded when TwinCAT is started.



Currently two tool types are supported:

- Drills
- Shaft Cutters

The relevant columns (parameters) for this type of tool are described below.

Drills

Parameter	Meaning
0	Tool number When this D-word is called, a tool number that is specified here can be given at the same time.
1	Tool type The drill is type 10 .
2	Geometry: Length Describes the length of the drill.

Parameter	Meaning
5	Wear: Length Describes the wear on the drill. The wear has to be given as a negative value, since it is added to the length.
8	Cartesian tool displacement [▶ 180] in X direction
9	Cartesian tool displacement in Y direction
10	Cartesian tool displacement in Z direction

Shaft Cutters

Parameter	Meaning
0	Tool number When this D-word is called, a tool number that is specified here can be given at the same time.
1	Tool type The shaft cutter is type 20
2	Geometry: Length Length of the shaft cutter.
4	Geometry: Radius
5	Wear: Length
7	Wear: Radius
8	Cartesian tool displacement [▶ 180] in X direction
9	Cartesian tool displacement in Y direction
10	Cartesian tool displacement in Z direction

Writing of tool data

Editing tool data with the XAE

As already mentioned, the tool data can be written directly from the XAE. To do this, edit the window shown above.

Parameterization of tool data via the PLC

In addition, tool data can be read and written from the PLC with the function block [ltpWriteToolDescEx](#) [[▶ 246](#)].

Writing tool data from the parts program

In some applications, it is more convenient to write the tool data directly from the part program.

The tool set to be overwritten must not be active during the write process. This means, for example, if tool radius compensation with parameter set D10 is active, this cannot be overwritten, as long as D10 is still selected.

Command	<code>#set ToolParam(<Zeile>; <Spalte>;<Wert>)#</code>
Parameter <line>	Describes the tool parameter line (1..255) Corresponds to the D number
Parameter <column>	Column to be written (0..15)
Parameter <value>	Parameter value to be transmitted

Sample:

```
N10 G0 X0 Y0 Z0
N20 G01 X100 F60000
N30 R1=10 R2=4 R3=20.3
N40 #set ToolParam(10; 0; 5)# #set ToolParam(10;1;20)#
N50 #set ToolParam(R1; R2; R3)#
N60 G41 X200 Y D10
...
```

Note No formulas may be transmitted as parameters. Writing of the tool data does not require a decoder stop.

Reading tool data from the parts program

This command can be used to assign tool data to an R-parameter.

Command	#get ToolParam(<line>; <column>;<R-Param>)#
Parameter <line>	Writes to the tool parameter line (1..255); this corresponds to the D number
Parameter <column>	Column to be written (0..15)
Parameter <R-Param>	R-parameter in which the date is entered

Sample:

```
N10 G0 X0 Y0 Z0
N20 G01 X100 F60000
N30 R1=10 R2=4
N40 #get ToolParam(10; 0; R5)# #getToolParam(10;1;R20)#
N50 #get ToolParam(R1; R2; R3)#
N60 G41 X200 Y D10
...
```

Notes:

Note No formulas may be transmitted as parameters. Reading of the tool data does not require a decoder stop.

5.4.2 Selecting and Deselecting the Length Compensation

Length compensation can only be selected when [G0 \[► 133\]](#) or [G1 \[► 134\]](#) are in effect. The [working plane \[► 126\]](#) must be selected to which the length compensation is perpendicular.

The feed direction is specified with P (see [working plane and feed direction \[► 126\]](#)).

To effect the movement corresponding to the length compensation, the axis concerned must at least be mentioned.

Sample:

```
N10 G17 G01 X0 Y0 Z0 F6000
N20 D1 X10 Y10 Z
N30 ...
N90 M30
```

Note Length correction is automatically selected when [cutter radius compensation \[► 183\]](#) is selected. To deselect length correction, D0 has to be programmed. It is again here necessary to at least mention the axis concerned in order to move to the new position.

5.4.3 Cartesian Tool Translation

Cartesian tool displacement refers to an offset between the reference point of the tool carrier and the reference point of the tool itself. In many cases, these reference points have the same location, so that a 0 can be entered for the tool displacement.

Parameter

The parameters for a translation are entered into the [tool data \[► 177\]](#) in the same way as the tool length etc. Parameters 8 to 10 are available for this purpose. Here

- P8 always describes the X-component
- P9 always describes the Y-component
- P10 always describes the Z-component

independently of the choice of level.

	TNr.(P0)	Typ(P1)	Geom.(P2)	Geom.(P3)
D 1	1	20	5.000000	0.00
D 2	0	0	0.000000	0.00
D 3	3	20	4.000000	0.00
D 4	0	0	0.000000	0.00
D 5	5	10	1.000000	0.00
D 6	0	0	0.000000	0.00

Selecting and deselecting Cartesian tool displacement

As in the case of length compensation, tool displacement is switched on with D<n> (n>0). In order to travel to the translated location, the axes must at least be named. This means that the displacement affects the positioning when the axis is called for the first time. It is also possible for a new final position to be entered for the axis.

The function is switched off with D0. Here again, it is necessary for the axes at least to be named, if the axes are to travel to their new co-ordinates.

Sample 1:

```
N10 G17 G01 X0 Y0 Z0 F6000
N20 D1 X10 Y10 Z (Z-Axis is repositioned)
N30 ...
N90 M30
```

Sample 2:

```
N10 G17 G01 X0 Y0 Z0 F6000
N20 D1 X10 Y10 (Z-Axis is not moved)
N30 ...
N90 M30
```

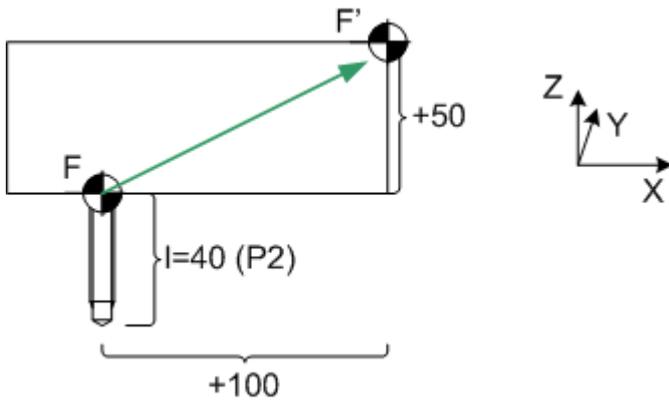
Using tool displacement and rotation



If the Cartesian tool displacement is used in combination with [rotation](#) [▶_145], then the compensation will only be correct if the aggregate (the tool carrier) is also rotated through the same angle.

Application sample

It often happens that a processing machine's tool carrier contains a number of tools. The appropriate tool is pneumatically activated according to the kind of machining required. Since, however, the tools are located at different positions, Cartesian tool displacement is required.



Tool parameters

Parameter	Value
0	0..65535
1	10
2	40
5	0
8	100.0
9	0.0
10	50

Behavior with incremental dimension notation

Default behavior

If a new tool offset (and also length compensation) is selected in incremental dimensions (G91), then the compensation is applied once the axis is named.

Sample 3:

```
(Tooloffset D1: X10 Y20 Z30)
N10 G01 D1 X100 Y0 Z0 F6000
N20 G91 (incremental dimension)
N30 D2 (Tooloffset D2: X100
Y200 Z300)
N30 Z10
N40 ...
```

Command	Description
ToolOffsetIncOn	The tool displacements and length compensations are also applied under G91 once the axis is named. (standard setting)
ToolOffsetIncOff	The tool displacement and length compensation are not applied under G91.

Sample 4:

```
(Tooloffset D1: X10 Y20 Z30)
N05 ToolOffsetIncOff
N10 G01 D1 X100 Y0 Z0 F6000
N20 G91 (incremental dimension)
N30 D2 (Tooloffset D2: X100
Y200 Z300)
N30 Z10
N40 ...
```

In N10 the Tooloffset is applied to all 3 axes. I.e. the axes move in the machine coordinate system (MCS) to X110 Y10 Z30.

In N30 the new Tooloffset of the Z-axis is **not** applied. This results in MCS X110 Y10 **Z40**.

See also [ZeroShiftIncOn/Off \[► 139\]](#)

5.4.4 Cutter Radius Compensation

5.4.4.1 Miller/Cutter Radius Compensation Off

Miller/Cutter Radius Compensation Off

Command	G40 (standard setting)
Cancellation	G41 [▶ 183] or G42 [▶ 184]

The G40 function switches the miller/cutter radius compensation off. The [length radius compensation \[▶ 180\]](#) will still remain active until it is switched off with D0. Between G40 and end of program it is mandatory to program at least one geometry element.

5.4.4.2 Miller/cutter radius compensation left

Miller/cutter radius compensation left

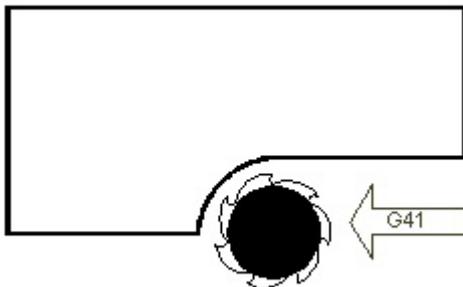
Command	G41
Cancellation	G40 [▶ 183]

The function G41 switches on the miller/cutter radius compensation. The tool is located to the **left** of the workpiece in the direction of movement.

As has already been seen for the [length compensation \[▶ 180\]](#), the cutter radius compensation can only be activated when [G0 \[▶ 133\]](#) or [G1 \[▶ 134\]](#) is in effect. The axes of the plane must be driven when the cutter radius compensation is selected.

Sample:

```
N10 G17 G01 X0 Y0 Z0 F6000
N20 G41 X10 Y20 Z D1
N30 X30
N40 G40 X10 Y10 Z
N50 M30
```



i Cutter radius compensation does not apply to full circles

The cutter radius compensation does not support full circles. Full circles have to be split into semi-circles, for sample.

Notes:

- The cutter radius compensation should be deactivated before the end of the NC program, in order to close it properly. Between G40 and end of program it is mandatory to program at least one geometry element.
- If a [decoder stop \[▶ 166\]](#) is programmed, cutter radius compensation has to be disabled first.
- For arcs, radius compensation can lead to a change in the path velocity at the contour. See also '[Path velocity in arcs \[▶ 188\]](#)'.

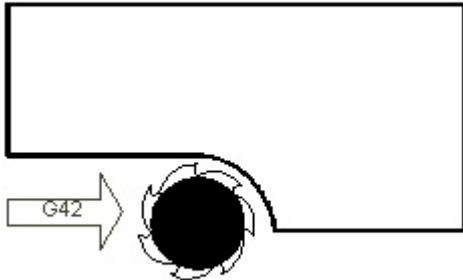
- See [Orthogonal contour approach/departure \[▶ 188\]](#).

5.4.4.3 Miller/cutter radius compensation right

Miller/cutter radius compensation right

Command	G42
Cancellation	G40 [▶ 183]

The function G42 switches on the miller/cutter radius compensation. The tool is located to the **right** of the workpiece in the direction of movement.



Cutter radius compensation does not apply to full circles

The cutter radius compensation does not support full circles. Full circles have to be split into semi-circles, for sample.

Note If a change is to be made from G41 to G42, then a G40 should be programmed between the two movements.

5.4.4.4 Departure and approach behavior of the miller/cutter radius compensation

This chapter describes the approach and departure behavior when the miller/cutter radius compensation is switched on or off. This behavior depends on the start position and cannot be influenced in any other way.

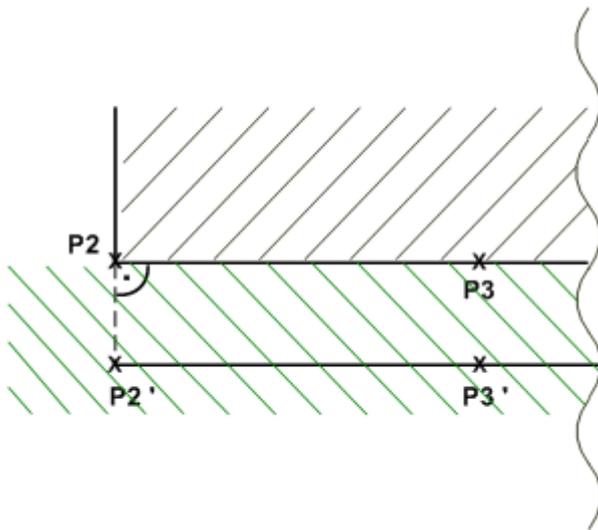
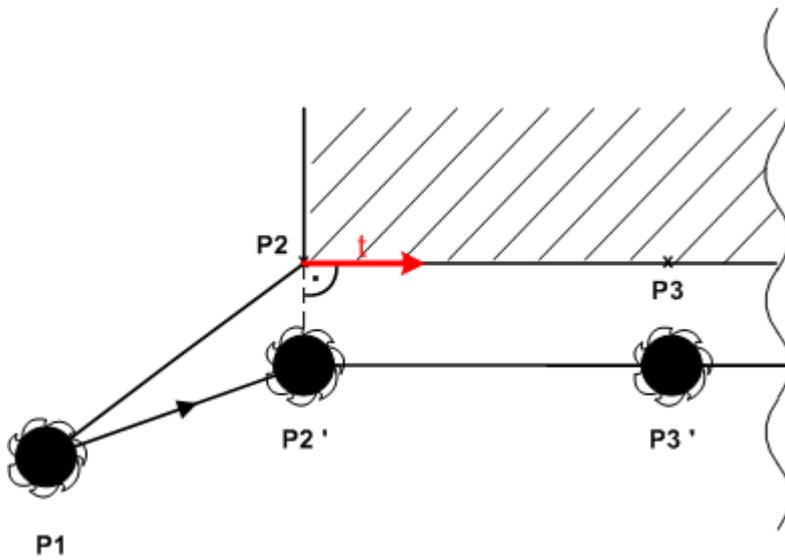
After the radius compensation is switched on, it must be applied. This means the cutter is at one point P1 (without radius compensation) and travels to P2', with the cutter radius being compensated at point P2'.

Point P2' depends on the start position P1 within the plain. A distinction is made between 3 basic cases. These cases are exemplified below during application of the radius compensation with a programmed G42 (right compensation).

Similar rules apply for the deactivation of the compensation, except that the tangent t is determined at the end of the path segment, with similar conditions being derived.

Case 1: P1 to the right of the path tangent t

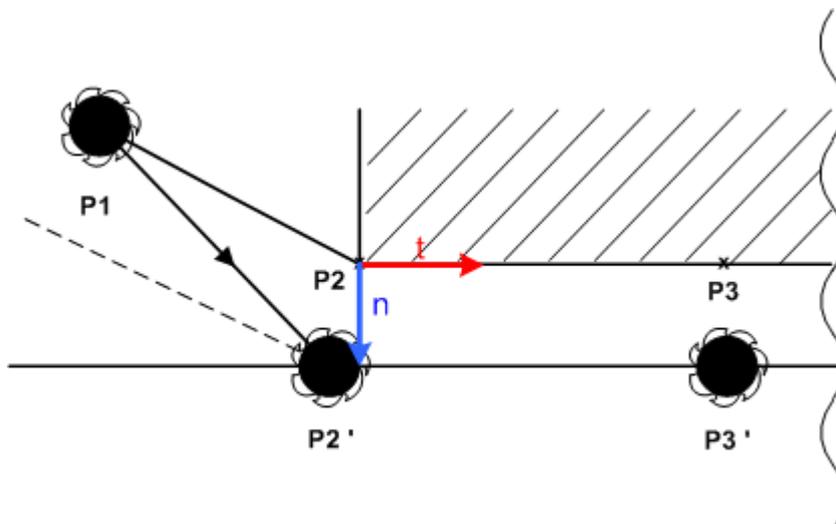
If the starting point P1 is to the right of the path tangent t , P2' is orthogonal to the tangent. This start-up behavior applies to the range hatched in green.

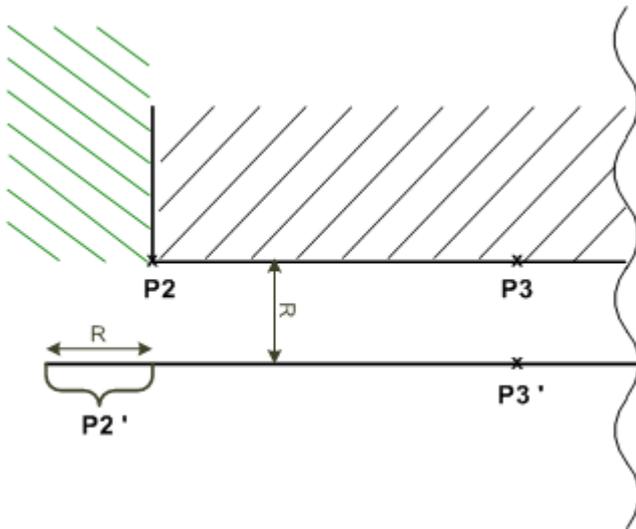


Case 2: P1 the right of the normal n and to the left of the path tangent t

If the start position P1 is to the right of the normal n and to the left of the path tangent t , P2' is moved. P2' results from the intersection of the parallel of P1P2 and the offset distance P2P3. Both straight lines are offset by radius R.

This behavior applies to the range hatched in green.

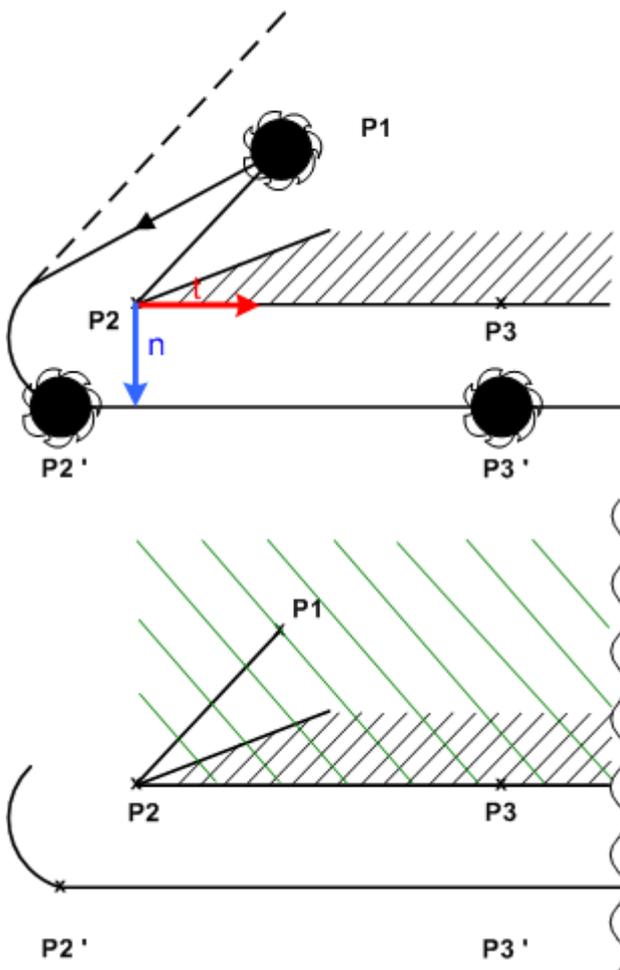




Case 3: P1 to the left of the normal n and to the left of the path tangent t

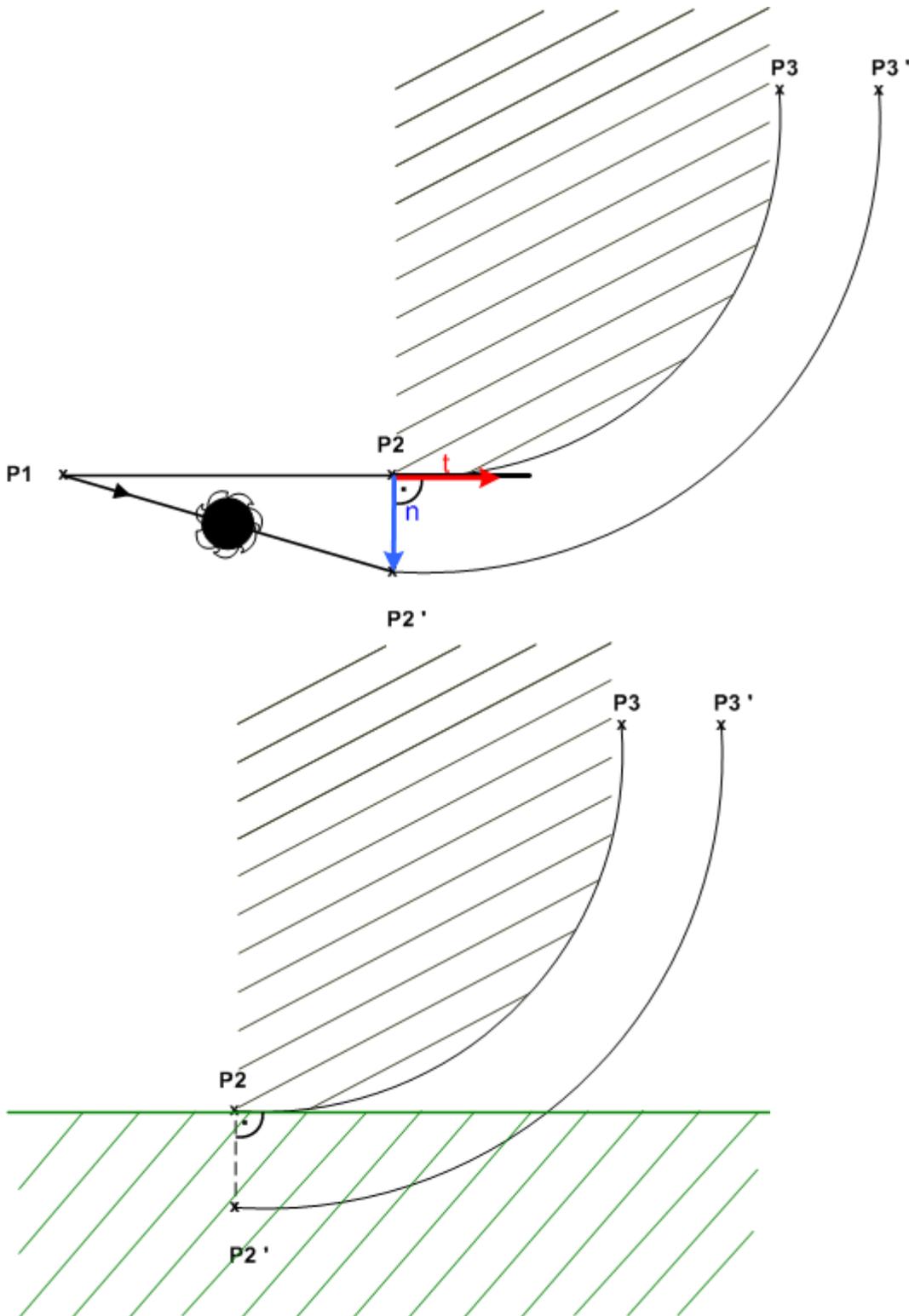
If the start position P1 is to the left of the normal n and also to the left of the path tangent t, an additional circle segment is inserted during approaching of P2'. In order to avoid free cutting at P2, P2' is not orthogonal to the start tangent of the section P2P3.

The additional circle segment is inserted for all start positions within the hatched green region.



A circle segment follows after the offset

The radius compensation is invariably applied via a straight line. (This must be set in the part program, since otherwise a runtime error will be generated). The contour can then start with a circle. The rules for starting and stopping are the same as before, i.e. here too the path tangent of the contour for P2 is determined, and a distinction is made between the three cases described.



If P2' is always to be approached orthogonal to the path tangent of P2, independent of the starting point, this can be realized with an additional command (see [Orthogonal contour approach/departure \[► 188\]](#)).

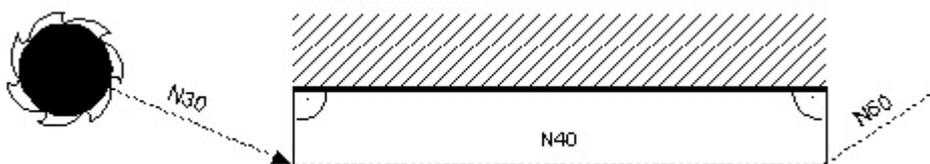
5.4.5 Orthogonal Contour Approach/Departure

Command	NORM
Cancellation	End of block
Programmable with	G40 ▶ 183 G41 ▶ 183 G42 ▶ 183

The 'NORM' command has the effect that the contour is approached orthogonally when cutter radius compensation is switched on. The actual position of the cutter is irrelevant. When de-selecting, the last segment with active compensation is also left orthogonally.

Sample:

```
N10 G17
N20 G01 X0 Y0 Z0 F6000
N30 G42 NORM X100 Y0 D5
N40 X200
N50 G40 NORM X220 Y0
N60 M30
```



Note The Norm command has hitherto only been implemented for straight line/straight line transitions.

5.4.6 Path Velocity in Arcs

When the cutter radius compensation |▶ 183| is active, the programmed radius changes for arcs. This in turn alters the velocity. The following commands are used to specify whether the feed value refers to the contour or the tool center point.

Constant Feed at the Contour

Command	CFC (standard setting)
Cancellation	CFIN or CFTCP

With CFC (constant feed contour) the feedrate at the contour is held constant.

Constant Feed at the Internal Radius

Command	CIN
Cancellation	CFC or CFTCP

With CFIN (constant feed internal radius) the feedrate at internal radii is reduced. This results in a constant velocity at the contour. The velocity at external radii is not increased.

Constant Feed of the Tool Centre Point

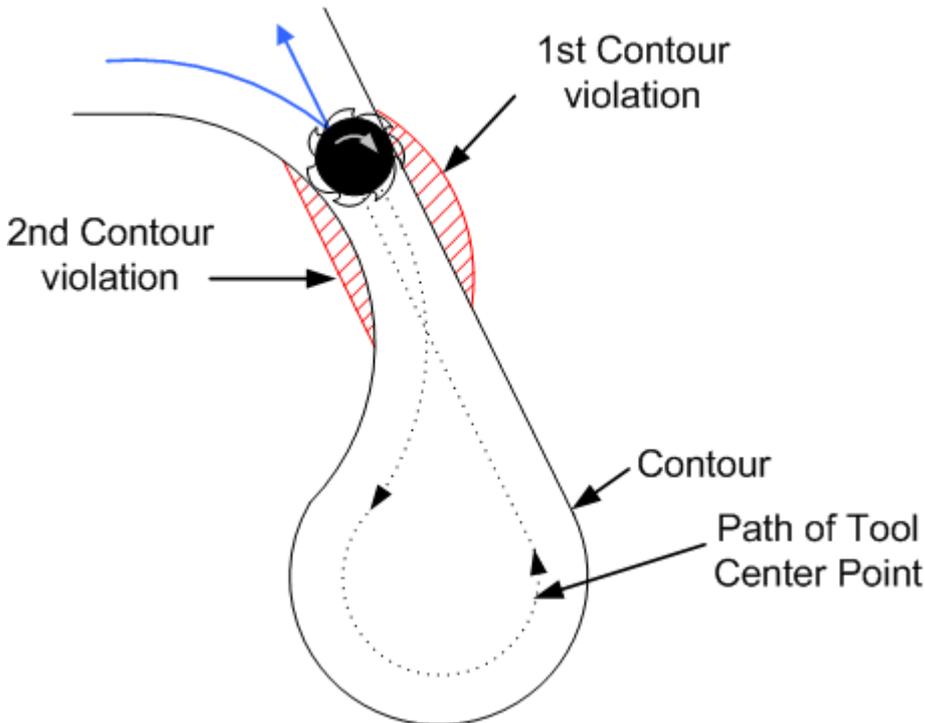
Command	CFTCP
Cancellation	CFC or CFIN

With CFTCP (constant feed tool center point) the feedrate of the tool's center point is kept constant. This means that at internal radii the velocity at the contour is increased, and that it is correspondingly reduced at external radii.

5.4.7 Bottle Neck Detection

If the cutter radius is not considered when a part program is created, the cutter may inadvertently process the opposite side of the workpiece. This leads to a contour collision with the workpiece, or, in other words, a bottleneck was programmed.

Command	CDON
Cancellation	CDOF



In this form, this behavior can only occur in combination with cutter radius compensation (G41/G42). In order to prevent such contour collisions, monitoring can be switched on from the part program via **CDON**. For it to be active, cutter radius compensation must also be selected.

The response of the NCI when a bottleneck is detected can be parameterized via the PLC. 3 cases are distinguished:

- Error and abort
If a bottleneck is detected, TwinCAT generates a runtime error and aborts the program execution.
- Notification and modification of the contour
If a bottleneck is detected, the contour is modified such that a contour collision is avoided (see Figure 1: blue line). However, this also means that segments may be left out, depending on the program. Furthermore, a note is entered in the application viewer to say that a bottleneck was detected.
- Notification and contour collision
If a bottleneck is detected, the contour is not changed and no error is generated. Only a message is entered in the application viewer.

Significant computing resources are required for contour collision monitoring. It should therefore only be selected if it is actually required. Furthermore, the amount of look-ahead for the bottleneck detection should be specified. This requires the number of future segments to be determined that are monitored relative to the n-th segment, in order to check for bottlenecks. The selected number of segments should not be too large, since this would put unnecessary strain on the system. The value for the look-ahead is also parameterized from the PLC.

Function blocks for parameterizing the bottleneck detection:

- [ItpSetBottleNeckModeEx \[► 234\]](#)
- [ItpGetBottleNeckModeEx \[► 210\]](#)
- [ItpSetBottleNeckLookAheadEx \[► 233\]](#)

- [ItpGetBottleNeckLookAheadEx \[► 209\]](#)

Sample:

```
N10 G0 X0 Y0 Z0
N20 CDON
N30 G01 G41 D3 X100 F6000 (cutter radius 30mm)
...
N40 G01 X200
N50 G02 X220 Y-74.641 I0 J-40
N60 G01 X300 Y-104
N70 G01 X230 Y120
N80 G40 D0 Y200
N90 CDOF
...
M30
```

5.5 Command overview

5.5.1 General command overview

Command	Description	block-by-block / modal	Default
ANG [► 144]	Contour line programming (angle)	s	
AROT [► 145]	Rotation additive	m	
CalcInvRot [► 145]	Calculates the inverse rotation of a vector	s	
CalcRot [► 145]	Calculates the rotation of a vector	s	
CDOF [► 189]	Bottleneck detection off	m	Default
CDON [► 189]	Bottleneck detection on	m	
CFC [► 188]	Constant velocity at the contour	m	Default
CFIN [► 188]	Constant velocity in the interior circle	m	
CFTCP [► 188]	Constant velocity of tool center point	m	
CIP [► 135]	Circular interpolation	s	
CPCOF [► 135]	Centre point correction off	m	
CPCON [► 135]	Centre point correction on	m	Default
DelDTG [► 156]	Delete Distance to Go	s	
DYNQVR [► 172]	Dynamic Override	m	
FCONST [► 138]	Constant feed programming	m	Default
FLIN [► 138]	Linear feed programming	m	
G00 [► 133]	Rapid traverse	m	
G01 [► 134]	Straight line interpolation	m	Default
G02 [► 135]	Clockwise circular interpolation	m	
G03 [► 135]	Anticlockwise circular interpolation	m	
G04 [► 137]	Dwell time	s	
G09 [► 138]	Accurate stop	s	

Command	Description	block-by-block / modal	Default
G17 [▶ 126]	Plane selection XY	m	Default
G18 [▶ 126]	Plane selection ZX	m	
G19 [▶ 126]	Plane selection YZ	m	
G40 [▶ 183]	No miller/cutter radius compensation	m	Default
G41 [▶ 183]	Miller/cutter radius compensation left	m	
G42 [▶ 183]	Miller/cutter radius compensation right	m	
G53 [▶ 139]	Zero shift suppression	m	Default
G54 [▶ 139]	1st adjustable zero shift	m	
G55 [▶ 139]	2nd adjustable zero shift	m	
G56 [▶ 139]	3rd adjustable zero shift	m	
G57 [▶ 139]	4th adjustable zero shift	m	
G58 [▶ 139]	1st programmable zero shift	m	
G59 [▶ 139]	2nd programmable zero shift	m	
G60 [▶ 138]	Accurate stop	m	
G70 [▶ 128]	Dimensions inch	m	
G71 [▶ 128]	Dimensions metric	m	Default
G74 [▶ 133]	Programmed traverse to reference point	s	
G90 [▶ 126]	Reference dimension notation	m	Default
G91 [▶ 126]	Incremental dimension notation	m	
G700 [▶ 128]	Dimensions in inches with calculation of the feed	m	
G710 [▶ 128]	Dimensions metric with calculation of the feed	m	
Mirror [▶ 148]	Mirroring coordinate system	m	
MOD [▶ 156]	Modulo movement	s	
MSG [▶ 177]	Messages from the NC program	s	
NORM [▶ 188]	orthogonal approach of and departure from the contour	s	
P+ [▶ 126]	Feed direction positive	m	Default
P- [▶ 126]	Feed direction negative	m	
paramAutoAccurateStop [▶ 155]	Automatic Accurate Stop	m	
paramAxisDynamics [▶ 172]	Parameterization of the axis dynamics	m	
paramC1ReductionFactor [▶ 173]	C1 reduction factor	m	
paramC2ReductionFactor [▶ 173]	C2 reduction factor	m	

Command	Description	block-by-block / modal	Default
paramCircularSmoothing [▶ 154]	Circular smoothing	m	
paramDevAngle [▶ 173]	C0 reduction - deflection angle	m	
paramGroupVertex [▶ 154]	Circular smoothing (old)	m	
paramGroupDynamic [▶ 172]	Pathway dynamics (old)	m	
paramPathDynamics [▶ 172]	Pathway dynamics	m	
paramRadiusPrec [▶ 136]	Circular accuracy	m	
paramSplineSmoothing [▶ 152]	Smoothing with Bezier Splines	m	
paramVertexSmoothing [▶ 149]	Smoothing of Segment Transitions	m	
paramVeloJump [▶ 173]	C0 reduction - max. step change in velocity	m	
paramVeloMin [▶ 175]	Minimum velocity	m	
paramZeroShift [▶ 139]	Parameterization of the configurable zero shift	m	
PathAxesPos [▶ 176]	Reads the actual position	s	
ROT [▶ 145]	Absolute rotation	m	
RotExOff [▶ 145]	Extended rotation function off	m	Default
RotExOn [▶ 145]	Extended rotation function on	m	
RotVec [▶ 145]	Calculation routine for rotating a vector	s	
RParam [▶ 130]	Initialization of R-parameters	s	
RToDwordGetBit [▶ 130]	Converts an R-parameter to DWord and checks whether a defined bit is set	m	
SEG [▶ 144]	Contour line programming (segment length)	s	
skip VirtualMovements [▶ 177]	Skip virtual movements	m	
ToolOffsetIncOff [▶ 180]	Cartesian tool displacement and length compensation is not applied under G91	m	
ToolOffsetIncOn [▶ 180]	Cartesian tool displacement and length compensation is applied under G91	m	Default
ToolParam [▶ 177]	Writing and reading of tool parameters	m	
TPM [▶ 142]	Target position monitoring	s	
ZeroShiftIncOff [▶ 139]	Zero shift is not applied under G91	m	

Command	Description	block-by-block / modal	Default
ZeroShiftIncOn [▶ 139]	Zero shift is applied under G91	m	Default

Address	Description
Q<n> [▶ 158]	Axis label for auxiliary axis (1 <= n <= 5)

5.5.2 @-Command Overview

Several variations of these commands are often possible, since K for a constant, R for an R-parameter or P for an R-parameter used as a pointer can be used for parameters. For example, the notation K/R/Pn should be understood to mean "either a number or an R-parameter or a pointer".

The following @-commands are available:

Command	Versions	Function
@40 [▶ 130]	@40 Kn Rn Rm	Save register on the stack
@41 [▶ 130]	@41 Rn Rm	Save register on the stack
@42 [▶ 130]	@42 Kn Rm Rn	Restore register from stack
@43 [▶ 130]	@43 Rm Rn	Restore register from stack
@100 [▶ 167]	@100 K±n @100 Rm	Unconditional jump
@111 [▶ 167]	@111 Rn K/Rn Km ...	Case block
@121 [▶ 167]	@121 Rn K/Rn Kn	Jump if unequal
@122 [▶ 167]	@122 Rn K/Rn Kn	Jump if equal
@123 [▶ 167]	@123 Rn K/Rn Kn	Jump if less or equal
@124 [▶ 167]	@124 Rn K/Rn Kn	Jump if less
@125 [▶ 167]	@125 Rn K/Rn Kn	Jump if greater or equal
@126 [▶ 167]	@126 Rn K/Rn Kn	Jump if greater
@131 [▶ 169]	@131 Rn K/Rn Kn	Loop while equal
@132 [▶ 169]	@132 Rn K/Rn Kn	Loop while unequal
@133 [▶ 169]	@133 Rn K/Rn Kn	Loop while greater
@134 [▶ 169]	@134 Rn K/Rn Kn	Loop while greater or equal
@135 [▶ 169]	@135 Rn K/Rn Kn	Loop while less
@136 [▶ 169]	@136 Rn K/Rn Kn	Loop while less or equal
@141 [▶ 169]	@141 Rn K/Rn Kn	Repeat until equal
@142 [▶ 169]	@142 Rn K/Rn Kn	Repeat until unequal
@143 [▶ 169]	@143 Rn K/Rn Kn	Repeat until greater
@144 [▶ 169]	@144 Rn K/Rn Kn	Repeat until greater or equal
@145 [▶ 169]	@145 Rn K/Rn Kn	Repeat until less
@146 [▶ 169]	@146 Rn K/Rn Kn	Repeat until less or equal
@151 [▶ 169]	@151 Rn K/Rn Kn	FOR_TO loop
@161 [▶ 169]	@161 Rn K/Rn Kn	FOR_DOWNTO loop
@200	@200 Rn	Delete a variable
@202	@202 Rn Rm	Swap two variables
@302	@302 K/R/Pn K/R/Pn R/Pn	Read machine data bit
@361 [▶ 176]	@361 Rn Km	Read machine-related actual axis value

Command	Versions	Function
@372	@372 Rn	Extract the NC-Channel-ID and store it in a variable
@402 [▶ 135]	@402 K/R/Pn K/R/Pn K/R/Pn	Write machine data bit
@610	@610 Rn Rn	Find absolute value of a variable
@613	@613 Rn Rn	Find square root of a variable
@614	@614 Rn Rm Rm	Find square root of the sum of the squares of two variables $x = \sqrt{a^2 + b^2}$
@620 [▶ 169]	@620 Rn	Increment variable
@621	@621 Rn	Decrement variable
@622	@622 Rn	Find integer part of a variable
@630 [▶ 131]	@630 Rn Rm	Find sine of a variable
@631 [▶ 131]	@631 Rn Rm	Find cosine of a variable
@632 [▶ 131]	@632 Rn Rm	Find tangent of a variable
@633 [▶ 131]	@633 Rn Rm	Find cotangent of a variable
@634 [▶ 131]	@634 Rn Rm	Find arcsine of a variable
@635 [▶ 131]	@635 Rn Rm	Find arccosine of a variable
@636 [▶ 131]	@636 Rn Rm	Find arctangent of a variable
@714 [▶ 166]	@714	Decoder stop
@716 [▶ 166]	@716	Decoder stop with rescan of the axis positions
@717 [▶ 166]	@717	Decoder stop with external trigger event

Machine data

Access to the following machine data is supported:

Byte	Bit	Action
5003 [▶ 135]	5	0: IJK words specify the distance between the center of the circle and the starting point. 1: IJK are absolute data giving the center of the circle.

6 PLC NCI Libraries

Requirements

Overview of PLC NCI libraries	Description
PLC Library: Tc2_NCI [▶ 195]	Function blocks for the configuration of the interpolation group (e.g. formation of the 3D group) and for operating the interpreter (G-Code (DIN 66025)) such as loading and starting the NC program.
PLC Library: Tc2_PlcInterpolation [▶ 294]	Programming of multi-dimensional movements from the PLC (alternative to using G-Code (DIN 66025))

6.1 PLC Library: Tc2_NCI

6.1.1 Configuration

The library Tc2_NCI provides function blocks for general NC axis configuration. This makes it possible to configure or to reconfigure axes in a simple way directly from the PLC.

Function Block	Description
CfgBuild3DGroup [▶ 195]	Groups up to 3 PTP axes into a 3D group
CfgBuildExt3DGroup [▶ 196]	Groups up to 3 PTP axes and 5 auxiliary axes into a 3D group
CfgAddAxisToGroup [▶ 198]	Configures a single axis at a particular location within a group (PTP, 3D, FIFO)
CfgReconfigGroup [▶ 198]	Removes 3D (or FIFO) axis allocations and returns of the axes to their personal PTP group
CfgReconfigAxis [▶ 199]	Returns a single axis from, for example, a 3D group, to its personal PTP group
CfgRead3DAxisIds [▶ 200]	Reads the axis IDs (axis allocation) of a 3D group
CfgReadExt3DAxisIds [▶ 201]	Reads the axis IDs (axis allocation) of a 3D group with auxiliary axes

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.1 CfgBuild3DGroup



This function block configures a 3D group with up to 3 PTP axes (X, Y and Z).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGroupId      : UDINT;
  nXAxisId      : UDINT;
  nYAxisId      : UDINT;
  nZAxisId      : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGroupId: ID of the 3D group

nXAxisId: ID of the PTP axes

nYAxisId: ID of the PTP axes

nZAxisId: ID of the PTP axes

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId        : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.2 CfgBuildExt3DGroup



This function block configures a 3D group with up to 3 path axes (X, Y and Z). Additionally, up to 5 auxiliary axes (Q1..Q5) can be configured.

The axis IDs of the PTP axes that are to be included in the interpolation group are applied at the inputs **nXAxisId** to **nQ5AxisId**.

Note The assignment of the auxiliary axes must start with **nQ1AxisId**. No gaps between auxiliary axes are permitted. For example, if **nQ3AxisId** is to be assigned, **nQ2AxisId** must also be assigned a valid Axis ID.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGroupId      : UDINT;
  nXAxisId      : UDINT;
  nYAxisId      : UDINT;
  nZAxisId      : UDINT;
  nQ1AxisId     : UDINT;
  nQ2AxisId     : UDINT;
  nQ3AxisId     : UDINT;
  nQ4AxisId     : UDINT;
  nQ5AxisId     : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is executed when a positive edge is encountered.

nGroupId: ID of the 3D group

nXAxisId: Axis IDs of the PTP axes to be included in the interpolation group

nYAxisId: Axis IDs of the PTP axes to be included in the interpolation group

nZAxisId: Axis IDs of the PTP axes to be included in the interpolation group

nQ1AxisId: Axis IDs of the PTP axes to be included in the interpolation group

nQ2AxisId: Axis IDs of the PTP axes to be included in the interpolation group

nQ3AxisId: Axis IDs of the PTP axes to be included in the interpolation group

nQ4AxisId: Axis IDs of the PTP axes to be included in the interpolation group

nQ5AxisId: Axis IDs of the PTP axes to be included in the interpolation group

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId        : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.3 CfgAddAxisToGroup



The CfgAddAxisToGroup function block configures a single axis at a particular location within an existing group (PTP, 3D, FIFO).

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    nGroupId      : UDINT;
    nAxisId       : UDINT;
    nIndex        : UDINT;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGroupId: ID of the target group

nAxisId: ID of the axis to be configured

nIndex: Position of the axis within the group; can have values between 0 and n-1. Depending on the type of group, n has the following significance: PTP: n = 1, 3D: n = 3, FIFO: n = 8

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.4 CfgReconfigGroup



The CfgReconfigGroup function block removes the axis allocation of an existing group (NCI or FIFO), returning the axes to their personal PTP groups.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGroupId      : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGroupId: ID of the group to be resolved

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.5 CfgReconfigAxis



The CfgReconfigAxis function block returns a single axis from, for example, a 3D group, to its personal PTP group.

Interface

```
VAR_INPUT
  bExecute      : BOOL;
  nAxisId       : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nAxisId: ID of the axis to be returned

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.6 CfgRead3DAxisIds



The function block CfgRead3DAxisIds reads the axis configuration of a 3D group.

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  nGroupId    : UDINT;
  pAddr       : PVOID;
  tTimeOut    : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGroupId: ID of the 3D group

pAddr: Address of the variable into which the function block writes the axis IDs of the group assignment (array with three elements of type UDINT)

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Sample:

```
VAR
  (* instance *)
  ReadAxIds : CfgRead3DAxisIds;
  AxIds : ARRAY[1..3] OF UDINT;
END_VAR

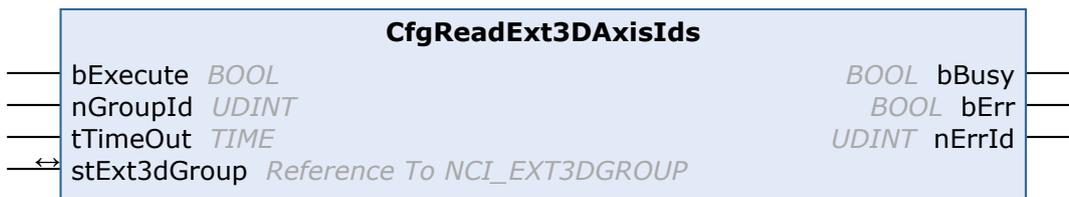
ReadAxIds( bExecute := TRUE,
  nGroupId := 4,
  pAddr := ADR( AxIds ),
  tTimeOut := T#1s );
```

AxIds now contains the three axis IDs for the 3D group with the group ID 4.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.1.7 CfgReadExt3DAxisIds



The function block CfgReadExt3DAxisIds reads the axis configuration of the extended 3D group.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGroupId      : UDINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGroupId: ID of the 3D group

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  stExt3dGroup : NCI_EXT3DGROUP;
END_VAR
```

stExt3dGroup: Instance of the structure NCI_EXT3DGROUP (enter axis IDs of the current interpolation group here)

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

```

TYPE NCI_EXT3DGROUP :
STRUCT
  nXAxisId      : UDINT;
  nYAxisId      : UDINT;
  nZAxisId      : UDINT;
  nQ1AxisId     : UDINT;
  nQ2AxisId     : UDINT;
  nQ3AxisId     : UDINT;
  nQ4AxisId     : UDINT;
  nQ5AxisId     : UDINT;
END_STRUCT
END_TYPE

```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2 NCI POU's

The TwinCAT library Tc2_NCI contains function blocks for operating the NC interpreter from the PLC.

The following function blocks are included in the library Tc2_NCI.

Function Block	Description
ltpConfirmHsk [► 204]	Acknowledges an M-function of type handshake
ltpDelDtgEx [► 205]	Triggers "Delete Distance to go" in the NC
ltpEnableDefaultGCode [► 206]	Executes a user-defined standard G-Code before the start of each NC program
ltpEStopEx [► 207]	Triggers the NCI EStop
ltpGetBlockNumber [► 208]	Provides the block number of the NC program for the cyclic interface
ltpGetBottleNeckLookAheadEx [► 209]	Provides the value of the look-ahead for bottleneck detection
ltpGetBottleNeckModeEx [► 210]	Provides the response mode for bottleneck detection
ltpGetChannelId [► 211]	Provides the channel ID
ltpGetChannelType [► 211]	Provides the channel type of the cyclic interface
ltpGetCyclicLRealOffsets [► 212]	Provides the index offset of the LREAL variables used in the cyclic channel interface
ltpGetCyclicUdintOffsets [► 213]	Provides the index offset of the UDINT variables used in the cyclic channel interface
ltpGetError [► 214]	Provides the error number
ltpGetGeoInfoAndHParamEx [► 215]	Reads information of the currently active segment and past and future segments.
ltpGetGroupAxisIds [► 216]	Provides the axis IDs that were configured for the group
ltpGetGroupId [► 217]	Provides the group ID

Function Block	Description
ItpGetHParam [▶ 217]	Provides the current H-parameter from the NC
ItpGetHskMFunc [▶ 218]	Provides the current M-function number of type handshake
ItpGetItfVersion [▶ 218]	Provides the current version of the cyclic interface
ItpGetOverridePercent [▶ 219]	Provides the channel override in percent
ItpGetSetPathVelocity [▶ 219]	Returns the current set path velocity
ItpGetSParam [▶ 220]	Provides the current S-parameter from the NC
ItpGetStateInterpreter [▶ 220]	Provides the current interpreter status
ItpGetTParam [▶ 221]	Provides the current T-parameter from the NC
ItpGoAheadEx [▶ 222]	Triggers the GoAhead function (decoder stop with external trigger event)
ItpHasError [▶ 223]	Determines whether an error is present
ItpIsFastMFunc [▶ 223]	Determines whether the M-function number provided is present in the form of a fast M-function
ItpIsEStopEx [▶ 224]	Determines whether an EStop is executed or pending
ItpIsHskMFunc [▶ 225]	Determines whether an M-function of type handshake is present
ItpLoadProgEx [▶ 225]	Loads an NC program using program names
ItpReadCyclicLRealParam1 [▶ 226]	Reads the first LReal parameter from the cyclic channel interface
ItpReadCyclicUdintParam1 [▶ 227]	Reads the first UDINT parameter from the cyclic channel interface
ItpReadRParamsEx [▶ 228]	Reads calculation parameters
ItpReadToolDescEx [▶ 229]	Reads the tool description from the NC
ItpReadZeroShiftEx [▶ 230]	Reads the zero shift from the NC
ItpResetEx2 [▶ 231]	Carries out a reset of the interpreter or of the NC channel
ItpResetFastMFuncEx [▶ 232]	Resets a fast signal bit
ItpSetBottleNeckLookAheadEx [▶ 233]	Sets the value of the look-ahead for bottleneck detection
ItpSetBottleNeckModeEx [▶ 234]	Sets the response mode when bottleneck detection is switched on
ItpSetCyclicLRealOffsets [▶ 235]	Sets the index offsets of the LREAL variables used in the cyclic channel interface
ItpSetCyclicUdintOffsets [▶ 236]	Sets the index offsets of the UDINT variables used in the cyclic channel interface
ItpSetOverridePercent [▶ 237]	Sets the channel override in percent
ItpSetSubroutinePathEx [▶ 238]	Optionally sets the search path for subroutines
ItpSetToolDescNullEx [▶ 239]	Sets all tool parameters (including number and type) to zero
ItpSetZeroShiftNullEx [▶ 240]	Set all zero shifts to null
ItpSingleBlock [▶ 241]	Enables or disables the single block mode in the NCI.
ItpStartStopEx [▶ 243]	Starts or stops the interpreter (NC channel)
ItpStepOnAfterEStopEx [▶ 244]	Enables further processing of the parts program after an NCI EStop
ItpWriteRParamsEx [▶ 245]	Writes calculation parameters
ItpWriteToolDescEx [▶ 246]	Writes the tool description into the NC
ItpWriteZeroShiftEx [▶ 247]	Writes the zero shift into the NC

Function Block	Description
Block search (for description of the functionality see Blocksearch [▶ 248])	
ItpBlocksearch [▶ 249]	Sets the interpreter to a user-defined location, so that NC program execution continues from this point.
ItpGetBlocksearchData [▶ 252]	Reads the current state after an NC program interruption.
ItpStepOnAfterBlocksearch [▶ 253]	Starts the motion after a block search.
Retrace	
ItpEnableFeederBackup [▶ 254]	Enables the backup list for retracing
ItpIsFeederBackupEnabled [▶ 255]	Reads whether the backup list for retracing is active
ItpIsFirstSegmentReached [▶ 256]	Reads whether the start position is reached during retracing
ItpIsFeedFromBackupList [▶ 256]	Reads whether feeder entries were sent from the backup list
ItpIsMovingBackwards [▶ 257]	Reads whether backward movement occurs on the current path
ItpRetraceMoveBackward [▶ 258]	Performs a backward movement on the path
ItpRetraceMoveForward [▶ 259]	Performs a forward movement on the path. This is called to cancel retracing.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.1 ItpConfirmHsk



The ItpConfirmHsk function block confirms the currently present M-function.

If the channel override is set to 0 or an E-stop is active, no M-functions are acknowledged during this time. The busy signal of ItpConfirmHsk therefore remains active and must continue to be called.

VAR_INPUT

```
VAR_INPUT
    bExecute : BOOL;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc : NCTOPLC_NCICHANNEL_REF;
    sPlcToNci : PLCTONC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

sPlcToNci: The structure of the cyclic channel interface from the PLC to the NCI. (Type: [PLCTONC_NCICHANNEL_REF](#) [▶ 325])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.2 ItpDelDtgEx



The function block ItpDelDtgEx triggers residual distance deletion. There is a more detailed description in the Interpreter [▶ 156]documentation.

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  tTimeOut    : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc   : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.3 ItpEnableDefaultGCode



The function block ItpEnableDefaultGCode enables execution of a user-defined G-Code before the start of each NC program from the PLC. The default program is executed before the loaded program when the actual NC program starts.

This function block enables rotation of the coordinate system for all NC programs to be executed, for example.

The standard G-Code must be stored as "DefaultGCode<Channel-Number>.def" in the TwinCAT\Mc\Nci directory.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    bUseDefaultGCode : BOOL;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

bUseDefaultGCode: If this variable is TRUE, the default G-Code is activated through a rising edge at bExecute. If the variable is FALSE, the default G-Code is deactivated.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy      : BOOL;
    bErr       : BOOL;
    nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).



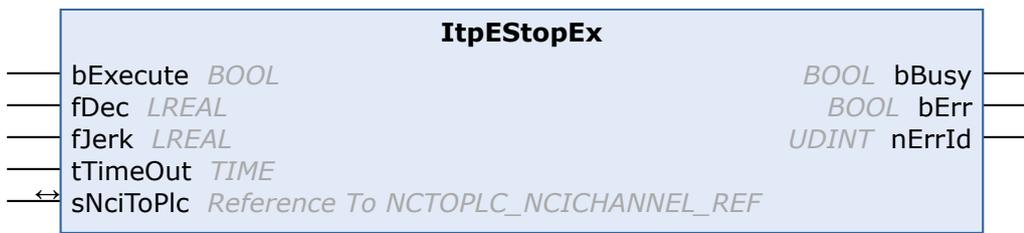
Not Available for GST

This function block is not available if the GST interpreter is employed.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.4 ItpEStopEx



The function block ItpEStopEx triggers the NCI EStop and enables a controlled stop on the path. The limit values for the deceleration and the jerk are transferred as parameters. If these should be smaller than the currently active dynamic parameters, the transferring parameters are rejected.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    fDec          : LREAL;
    fJerk        : LREAL;
    tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

fDec: Max. deceleration during stopping. If fDec is smaller than the currently active deceleration, fDec is rejected. This ensures that the deceleration occurs with the standard ramp as a minimum.

fJerk: Max. jerk during stopping. If fJerk is smaller than the currently active jerk, fJerk is rejected.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also [ItpStepOnAfterEStopEx](#) [▶ 244].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.5 ItpGetBlockNumber



ItpGetBlockNumber is a function that returns the block number of the NC program for the cyclic interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetBlockNumber: Block number of the active geometry segment

Sample

```
VAR
  nBlockNumber      : UDINT;
  sNciToPlc AT%I*   : NCTOPLC_NCICHANNEL_REF;
END_VAR

nBlockNumber := ItpGetBlockNumber(sNciToPlc);
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.6 ItpGetBottleNeckLookAheadEx



The function block ItpGetBottleNeckLookAheadEx determines the maximum size of the look-ahead for the bottleneck detection (contour collision monitoring).

There is a more detailed description in the [Interpreter \[► 189\]](#) documentation.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[► 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
    nLookAhead   : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. If the function block has a timeout error, 'Error' is TRUE and 'nErrId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

nLookAhead: Value of the look-ahead for bottleneck detection

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.7 ItpGetBottleNeckModeEx



The function block ItpGetBottleNeckModeEx reads the behavior in the event of a contour collision (bottleneck).

There is a more detailed description in the [Interpreter \[▶ 189\]](#) documentation.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    tTimeout      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

tTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[▶ 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy          : BOOL;
    bErr           : BOOL;
    nErrId         : UDINT;
    eBottleNeckMode : E_ItpBnMode
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

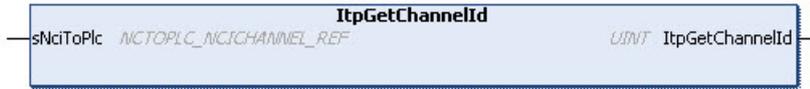
eBottleNeckMode: Enum for the behavior in the event of a contour collision

```
TYPE E_ItpBnMode:
(
    ItpBnm_Abort := 0,
    ItpBnm_Adjust := 1,
    ItpBnm_Leave := 2
);
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.8 ItpGetChannelId



ItpGetChannelId is a function that determines the channel ID from the cyclic interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetChannelId: Channel ID (type: UDINT)

Sample

```
VAR
  nChnId      : UDINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR

nChnId := ItpGetChannelId( sNciToPlc );
```

see also: [ItpGetGroupId](#) [▶ 217]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.9 ItpGetChannelType



ItpGetChannelType is a function that returns the channel type of the cyclic interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetChannelType: Channel type (type: E_ItpChannelType)

```
TYPE E_ItpChannelType :
(
  ItpChannelTypeNone,
  ItpChannelTypeInterpreter,
  ItpChannelTypeKinematic,
  ItpChannelType_InvalidItfVer := 16#4B14 (*ErrToNciItp_ItfVersion the cyclic channel interface does not match to the requested function/fb *)
);
END_TYPE
```

Sample

```

VAR
  nChannelType      : E_ItpChannelType;
  sNciToPlc AT%I*  : NCTOPLC_NCICHANNEL_REF;
END_VAR

nChannelType := ItpGetChannelType( sNciToPlc );
    
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.10 ItpGetCyclicLrealOffsets



The function block ItpGetCyclicLrealOffsets is used to read the current configuration of the cyclic channel interface for LREAL variables.

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  tTimeout      : TIME;
END_VAR
    
```

bExecute: the command is executed by a rising edge at this input.

tTimeout: ADS timeout delay

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
    
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy          : BOOL;
  bErr           : BOOL;
  nErrId         : UDINT;
  nIndexOffsetParam1 : UDINT;
  nIndexOffsetParam2 : UDINT;
  nIndexOffsetParam3 : UDINT;
  nIndexOffsetParam4 : UDINT;
END_VAR
    
```

bBusy: this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

nIndexOffsetParam1: group state ([index offset](#)) for parameter 1

nIndexOffsetParam2: group state ([index offset](#)) for parameter 2

nIndexOffsetParam3: group state ([index offset](#)) for parameter 3

nIndexOffsetParam4: group state ([index offset](#)) for parameter 4

See also:

- [ItpReadCyclicLRealParam1](#) [[▶ 226](#)]
- [ItpSetCyclicLRealOffsets](#) [[▶ 235](#)]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.11 ItpGetCyclicUDintOffsets



The function block ItpGetCyclicUDintOffsets is used to read the current configuration of the cyclic channel interface for UDINT variables.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    tTimeOut     : TIME;
END_VAR
```

bExecute: the command is executed by a rising edge at this input.

tTimeOut: ADS timeout delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [[▶ 323](#)])

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
    nIndexOffsetParam1 : UDINT;
    nIndexOffsetParam2 : UDINT;
```

```

nIndexOffsetParam3 : UDINT;
nIndexOffsetParam4 : UDINT;
END_VAR

```

bBusy: this output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: this output is switched to TRUE if an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

nIndexOffsetParam1: group state ([index offset](#)) for parameter 1

nIndexOffsetParam2: group state ([index offset](#)) for parameter 2

nIndexOffsetParam3: group state ([index offset](#)) for parameter 3

nIndexOffsetParam4: group state ([index offset](#)) for parameter 4

See also:

- [ItpReadCyclicUdintParam1](#) [[▶ 227](#)]
- [ItpSetCyclicUdintOffsets](#) [[▶ 236](#)]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.12 ItpGetError



ItpGetError is a function that returns the error number. A description of the NC error codes can be found [here](#).

VAR_IN_OUT

```

VAR_IN_OUT
sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR

```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [[▶ 323](#)])

Return value

ItpGetError: Error number



ItpGetError evaluates the variable 'nItpErrCode' from the cyclic interface.

Sample

```

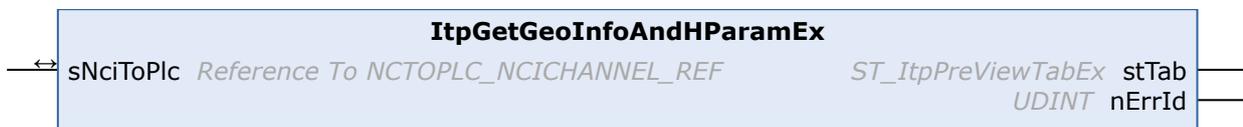
VAR
  bItpError      : BOOL;
  nErrId         : UDINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR

bItpError := ItpHasError( sNciToPlc );
IF bItpError THEN
  nErrId := ItpGetError( sNciToPlc );
  ...
END_IF
    
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.13 ItpGetGeoInfoAndHParamEx



The function block `ItpGetGeoInfoAndHParamEx` reads informations of the currently active segment and past and future segments. These include block number, H-parameter and residual path length on the segment.

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
    
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |▸ 323|)

VAR_OUTPUT

```

VAR_OUTPUT
  stTab          : ST_ItpPreViewTabEx;
  nErrId         : UDINT;
END_VAR
    
```

stTab: Structure containing the segment data. See [ST_ItpPreViewTabEx](#) |▸ 215|.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in `nErrId` can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

```

TYPE ST_ItpPreViewTabEx :
STRUCT
  nDcTime      : UDINT := 0;
  nReserved    : UDINT := 0;
  arrLines     : ARRAY[1..NCI_MAX_PREVIEWTABLINES] OF ST_ItpPreViewTabLine;
END_STRUCT
END_TYPE
    
```

nDcTime: Current time stamp in ns. This time stamp can be used e.g. in interplay with the `Tc2_NciXFC` library.

arrLines: Array of segment-related information (size 20). The entry at position 11 of the array corresponds to the currently active segment. Segments that have already been processed are displayed at positions 1-10 of the array, future segments at positions 12-20. See [ST_ItpPreViewTabLine](#) |▸ 215|.

```

TYPE ST_ItpPreViewTabLine :
STRUCT
  fLength      : LREAL := 0.0;
  nBlockNo     : UDINT := 0;
END_STRUCT
    
```

```
nHParam      : UDINT := 0;
nEntryID     : UDINT := 0;
nReserved    : UDINT := 0;
END_STRUCT
END_TYPE
```

fLength: Remaining segment length. For segments that are not yet active this corresponds to the total segment length. For past segments the distance moved since the end of the segment is specified.

nBlockNo: Block number programmed by the user

nHParam: Value of the H-parameter [[▶ 165](#)] that is active from the start of the next segment

nEntryID: Command ID generated by the system

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.14 ItpGetGroupAxisIds



ItpGetGroupAxisIds is a function that returns an array of axes IDs that were configured for the group.

VAR_IN_OUT

```
FUNCTION ItpGetGroupAxisIds
VAR_IN_OUT
    sNciToPlc AT%I* : NCTOPLC_NCICHANNEL_REF;
    nNciAxisIds   : ARRAY[1..8] OF DWORD;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: NCTOPLC_NCICHANNEL_REF [[▶ 323](#)])

sNciAxisIds: Array of axis IDs

Return value

ItpGetGroupAxisIds: Error number



ItpGetGroupAxisIds evaluates the variable 'nItpErrCode' from the cyclic interface.

Sample

```
VAR
    nNciAxisIds   : ARRAY[1..8] OF DWORD;
    sNciToPlc AT%I* : NCTOPLC_NCICHANNEL_REF;
    nVersionErr   : DWORD;
END_VAR
nVersionErr := ItpGetGroupAxisIds(nNciAxisIds, sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.15 ItpGetGroupId



ItpGetGroupId is a function that determines the group ID from the cyclic interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

Return value

ItpGetGroupId: Group ID

Sample

```
VAR
  nGrpId      : UINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR
nGrpId := ItpGetGroupId( sNciToPlc );
```

See also: [ItpGetChannelId](#) |> 211]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.16 ItpGetHParam



ItpGetHParam is a function that returns the current H-parameter.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

Return value

ItpGetHParam: H parameter



ItpGetHParam evaluates the variable 'nHFuncValue' from the cyclic interface.

Sample

```

VAR
  nHParam      : DINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR

nHParam := ItpGetHParam( sNciToPlc );

```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.17 ItpGetHskMFunc



ItpGetHskMFunc supplies the number of the M-function of type handshake.

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR

```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetHskMFunc: Number of the M-function



ItpGetHskMFunc evaluates the variable 'nHskMFuncNo' from the cyclic interface.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.18 ItpGetItfVersion



ItpGetItfVersion is a function that determines the version number of the cyclic interface.

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR

```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetItfVersion: Version number of the cyclic interface

Sample

```
VAR
  nItfVer      : UINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR

nItfVer := ItpGetItfVersion( sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.19 ItpGetOverridePercent



The ItpGetOverridePercent function returns the axis channel override as a percentage. It is essential to remember that this is not a value from the NC. The value, which is transferred as set value to the NC, is evaluated.

VAR_IN_OUT

```
VAR_IN_OUT
  sPlcToNci      : PLCTONC_NCICHANNEL_REF;
END_VAR
```

sPlcToNci: Structure of cyclic channel interface between PLC and NCI (type: [PLCTONC_NCICHANNEL_REF](#) [[▶ 325](#)])

Return value

ItpGetOverridePercent: Override in percent

Sample

```
VAR
  sPlcToNci AT%Q*: PLCTONC_NCICHANNEL_REF;
  fOverride      : LREAL;
END_VAR

fOverride := ItpGetOverridePercent( sPlcToNci );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.20 ItpGetSetPathVelocity



ItpGetSetPathVelocity is a function that reads the current set path velocity from the cyclic interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

Return value

ItpGetSetPathVelocity: Current set path velocity

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.21 ItpGetSParam



ItpGetSParam is a function that returns the current S-parameter.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

Return value

ItpGetSParam: S parameter

i ItpGetSParam evaluates the variable 'nSpindleRpm' from the cyclic interface.

Sample

```
VAR
  nSParam          : UINT;
  sNciToPlc AT%I* : NCTOPLC_NCICHANNEL_REF;
END_VAR

nSParam := ItpGetSParam( sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.22 ItpGetStateInterpreter



ItpGetStateInterpreter is a function that returns the interpreter status.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetStateInterpreter: Current interpreter status [▶ 15]



ItpGetStateInterpreter evaluates the variable 'nItpState' from the cyclic interface.

Sample

```
VAR
  nItpState      : UDINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR

nItpState := ItpGetStateInterpreter( sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.23 ItpGetTParam



ItpGetTParam is a function that returns the current T-parameter.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

ItpGetTParam: T parameter



ItpGetTParam evaluates the variable 'nTool' from the cyclic interface.

Sample

```
VAR
  nTParam        : UINT;
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
END_VAR

nTParam := ItpGetTParam( sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.24 ItpGoAheadEx



The ItpGoAheadEx function block may only be used in association with the decoder stop '@717' [▶ 166]. There is a more detailed description of this decoder stop in the [interpreter documentation](#) [▶ 122].

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy          : BOOL;
    bErr           : BOOL;
    nErrId         : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.25 ItpHasError



ItpHasError is a function that determines whether the interpreter is in an error state.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

Return value

If there is an error, the function returns TRUE.



ItpHasError evaluates the variable 'nItpErrCode' from the cyclic interface. If this value does not equal 0, TRUE is returned.

Sample

```
VAR
  bItpError      : BOOL;
  sNciToPlc AT%I* : NCTOPLC_NCICHANNEL_REF;
END_VAR

bItpError := ItpHasError( sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.26 ItpIsFastMFunc



ItpIsFastMFunc is a function that determines whether the fast M-function is set for the supplied M-function number.

VAR_IN

```
FUNCTION ItpIsFastMFunc
VAR_IN
  nFastMFuncNo : INT;
END_VAR
```

nFastMFuncNo: Number of the M-function that is to be checked.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

Return value

The function returns TRUE if the fast bit of the M-function is set.

i ItpIsFastMFunc evaluates the variable 'nFastMFuncMask' from the cyclic interface.

Sample

```
(*this enum is defined by the user *)
TYPE FastMFuncs:
(
  M10_CoolingFluidOn := 10, (*fast M-Funktion M10*)
  M11_CoolingFluidOff := 11,
  M12_FanOn := 12,
  M13_FanOff := 13
);
END_TYPE

VAR
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF
  enFastMFuncs : FastMFuncs;
  bTurnFanOn : BOOL;
END_VAR

bTurnFanOn := ItpIsFastMFunc( M12_FanOn, sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.27 ItpIsEStopEx



The function ItpIsEStopEx indicates whether an EStop command was triggered.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

If the return value is TRUE, the function was preceded by an EStop (e.g. ItpEStopEx). The flag does **not** provide information as to whether the axes have already stopped or are still on the braking ramp.

After execution of ItpStepOnAfterEStopEx, ItpIsEStopEx returns FALSE again.

i ItpIsEStopEx evaluates the cyclic interface.

see also:

[ItpEStopEx](#) [▶ 207]

[ItpStepOnAfterEStopEx](#) [▶ 244]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.28 ItpIsHskMFunc



ItpIsHskMFunc determines whether an M-function of type handshake is present.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

Return value

The function returns TRUE if an M-function of type handshake is present.

i ItpIsHskFunc evaluates the variable 'nHskMFuncReq' from the cyclic interface.

Sample

```
VAR
  bMFuncRequest : BOOL;
  sNciToPlc AT%I* : NCTOPLC_NCICHANNEL_REF;
END_VAR

bMFuncRequest := ItpIsHskMFunc( sNciToPlc );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.29 ItpLoadProgEx



VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  sPrg          : STRING(255);
  nLength       : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The function block reads the NC program when a rising edge is encountered

sPrg: Name of the NC program that is loaded

nLength: String length of the program name

tTimeOut: ADS Timeout-Delay

The NC program is looked up in directory "TwinCAT\Mc\Nci", if no further information is available. It is however also possible to give an absolute path.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Sample

```
VAR
  in_stItpToPlc AT %I*      : NCTOPLC_NCICHANNEL_REF;
  fbLoadProg               : ItpLoadProgEx;
  sProgramPath             : STRING (255) := 'TestIt.nc';
END_VAR

fbLoadProg(
  bExecute := TRUE,
  sPrg     := sProgramPath,
  nLength  := LEN(sProgramPath),
  tTimeOut := t#200ms,
  sNciToPlc := in_stItpToPlc
);
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.30 ItpReadCyclicLRealParam1



This function reads the first LREAL parameter from the cyclic channel interface. This parameter is configured previously with [ItpSetCyclicLRealOffsets](#) [▶ 235].

Parameter 2 to 4 are read via the same mechanism (e.g. [ItpReadCyclicLRealParam2](#)).

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

Parameter 1 of type LREAL.

See also:

- [ItpReadCyclicUdintParam1](#) [▶ 227]
- [ItpSetCyclicLRealOffsets](#) [▶ 235]
- [ItpGetCyclicLRealOffsets](#) [▶ 212]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.31 ItpReadCyclicUdintParam1



This function reads the first UDINT parameter from the cyclic channel interface. This parameter is configured previously with [ItpSetCyclicUdintOffsets](#) [▶ 236].

Parameter 2 to 4 are read via the same mechanism (e.g. [ItpReadCyclicUdintParam2](#)).

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

Return value

Parameter 1 of type UDINT.

See also:

- [ItpReadCyclicLRealParam1](#) [▶ 226]
- [ItpSetCyclicUdintOffsets](#) [▶ 236]
- [ItpGetCyclicUdintOffsets](#) [▶ 213]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.32 ItpReadRParamsEx



The ItpReadRParamsEx function block reads the NC's calculation parameters, also known as R-parameters. A more detailed description of the calculation parameters can be found [here \[► 130\]](#). A total of 1000 R-parameters are available, of which the first 900 (0..899) are local, so that they are only visible in the current NC channel. The other 100 (900..999) R-parameters are global, and are thus visible from anywhere in the NC.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  pAddr         : PVOID;
  nIndex        : DINT;
  nCount        : DINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: A rising edge starts the read operation.

pAddr: Address of the target variables of the data to be read. Data are written directly from the specified address, i.e. nIndex is not to be interpreted as offset from pAddr. The data are usually stored in an array of type LREAL, which has to be defined by the user.

nIndex: Describes the index of the R-parameter to be read from an NC perspective.

nCount: Number of R-parameters to be read

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[► 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy          : BOOL;
  bErr           : BOOL;
  nErrId         : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.33 ItpReadToolDescEx



The ItpReadToolDescEx function block reads the tool parameters for the supplied D-word.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nDNo          : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nDNo: D-word for which the tool parameters are to be read. nDNo can have values between 1 and 255.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc     : NCTOPLC_NCICHANNEL_REF;
  sToolDesc     : ToolDesc;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading.

sToolDesc: A structure into which the tool parameters of nDNo are written. The meaning of the parameters depends on the tool type, and can be found in the [tool data \[▶ 177\]](#). (type: [NCTOPLC_NCICHANNEL_REF \[▶ 323\]](#))

```
TYPE ToolDesc:
STRUCT
  nToolNumber   : UDINT; (*valid range from 0 .. 65535*)
  nToolType     : UDINT;
  fParam        : ARRAY [2..15] OF LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ItpWriteToolDescEx](#) [[▶ 246](#)]

[ItpSetToolDescNullEx](#) [[▶ 239](#)]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.34 ItpReadZeroShiftEx



The ItpReadZeroShiftEx function block reads the zero shift components X, Y and Z for the given zero shift.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nZsNo        : UDINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nZsNo: Number of the zero shift. G54 to G59 are zero shifts at the NC. The valid range of values for 'nZsNo' is therefore from 54 to 59.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
  sZeroShiftDesc : ZeroShiftDesc;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [[▶ 323](#)])

sZeroShiftDesc: The structure containing the components of the zero shift.

```
TYPE ZeroShiftDesc:
STRUCT
  fShiftX      : LREAL;
  fShiftY      : LREAL;
  fShiftZ      : LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).



For reasons of compatibility, there are two entries (coarse and fine) for each axis in each zero shift (e.g. G54). These two entries must be added together. This function block evaluates both the entries and adds them together automatically.

See also:

[ItpWriteZeroShiftEx](#) [▶ 247]

[ItpSetZeroShiftNullEx](#) [▶ 240]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.35 ItpResetEx2



The function block 'ItpResetEx2' executes a channel reset, which deletes all existing tables of the NC channel. In contrast to the outdated function block ItpReset, an active channel is stopped first, before the reset is executed. This simplifies programming in the PLC, since no explicit check is necessary to ascertain whether the axes are still in motion.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

tTimeOut: ADS timeout delay (the bBusy signal can be active for longer than tTimeOut)

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCCHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.36 ItpResetFastMFuncEx



The fast M-function [nMFuncNo](#) [▶ 161] is reset with a rising edge at input bExecute. In the event of the M-function not being available, **no** error is returned.

This function block represents an alternative to Auto-reset or reset with another M-function (reset list during parameterization of the M-function). For reasons of transparency, mixed resets using an M-function and this function block should be avoided.

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  nMFuncNo    : UINT;
  tTimeOut    : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nMFuncNo: Flying M-function that is to be reset

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc   : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

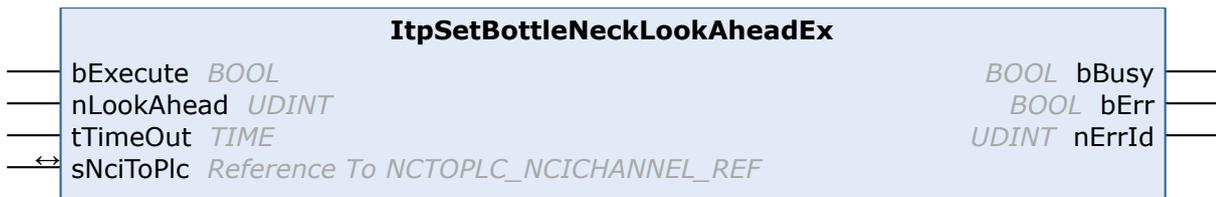
bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.37 ItpSetBottleNeckLookAheadEx



The function block ItpSetBottleNeckLookAheadEx determines the maximum number of segments the system may look ahead for bottleneck detection (contour collision monitoring). Note that segments, which were added as a result of radius compensation (e.g. additional segments at acute angles) are taken into account.

There is a more detailed description in the [Interpreter \[▶ 189\]](#) documentation.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nLookAhead    : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nLookAhead: Specifies the look-ahead value

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[▶ 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

● Not Available for GST
i This function block is not available if the GST interpreter is employed.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.38 ItpSetBottleNeckModeEx



The function block ItpSetBottleNeckModeEx specifies the behavior in the event of a contour collision (bottleneck).

There is a more detailed description in the [Interpreter \[► 189\]](#) documentation.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  eBottleNeckMode : E_ItpBnMode;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

eBottleNeckMode: Enum for the behavior in the event of a contour collision

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[► 323\]](#))

```

TYPE E_ItpBnMode:
(
  ItpBnm_Abort   := 0,
  ItpBnm_Adjust := 1,
  ItpBnm_Leave    := 2
);
END_TYPE
    
```

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy   : BOOL;
  bErr    : BOOL;
  nErrId  : UDINT;
END_VAR
    
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Not Available for GST
 This function block is not available if the GST interpreter is employed.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.39 ItpSetCyclicLrealOffsets



The function block ItpSetCyclicLrealOffsets is used to describe the cyclic channel interface for the 4 freely configurable LREAL variables. Variables (index offsets) can be selected from the group state.

The functionality is only active if nIndexOffsetParam1 is not equal 0.

VAR_INPUT

```

VAR_INPUT
  bExecute       : BOOL;
  tTimeOut       : TIME;
  nIndexOffsetParam1 : UDINT;
  nIndexOffsetParam2 : UDINT;
  nIndexOffsetParam3 : UDINT;
  nIndexOffsetParam4 : UDINT;
END_VAR
    
```

bExecute: the command is executed by a rising edge at this input.

tTimeOut: ADS timeout delay

nIndexOffsetParam1: group state ([index offset](#)) for parameter 1

nIndexOffsetParam2: group state ([index offset](#)) for parameter 2

nIndexOffsetParam3: group state ([index offset](#)) for parameter 3

nIndexOffsetParam4: group state ([index offset](#)) for parameter 4

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323|)

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy      : BOOL;
    bErr       : BOOL;
    nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

- [ItpReadCyclicLRealParam1](#) |> 226|
- [ItpGetCyclicLRealOffsets](#) |> 212|

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.40 ItpSetCyclicUDintOffsets



The function block ItpSetCyclicUDintOffsets is used to describe the cyclic channel interface for the 4 freely configurable UDINT variables. Variables ([index offsets](#)) can be selected from the group state.

The functionality is only active if nIndexOffsetParam1 is not equal 0.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  tTimeOut     : TIME;
  nIndexOffsetParam1 : UDINT;
  nIndexOffsetParam2 : UDINT;
  nIndexOffsetParam3 : UDINT;
  nIndexOffsetParam4 : UDINT;
END_VAR
```

bExecute: the command is executed by a rising edge at this input.

tTimeOut: ADS timeout delay

nIndexOffsetParam1: group state ([index offset](#)) for parameter 1

nIndexOffsetParam2: group state ([index offset](#)) for parameter 2

nIndexOffsetParam3: group state ([index offset](#)) for parameter 3

nIndexOffsetParam4: group state ([index offset](#)) for parameter 4

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | [323](#))

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

- [ItpReadCyclicUdintParam1](#) | [227](#)
- [ItpGetCyclicUdintOffsets](#) | [213](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.41 ItpSetOverridePercent



The function `ItpSetOverridePercent` writes the axes channel override into the cyclic interface of the NCI. The override is passed as a percentage.

VAR_INPUT

```
FUNCTION ItpSetOverridePercent
VAR_INPUT
    fOverridePercent : LREAL;
END_VAR
```

fOverridePercent: Axis channel override as a percentage

VAR_IN_OUT

```
VAR_IN_OUT
    sPlcToNci : PLCTONC_NCICHANNEL_REF;
END_VAR
```

sPlcToNci: Structure of cyclic channel interface between PLC and NCI (type: [PLCTONC_NCICHANNEL_REF](#) [[▶ 325](#)])

Return value

ItpSetOverridePercent: always TRUE

Sample

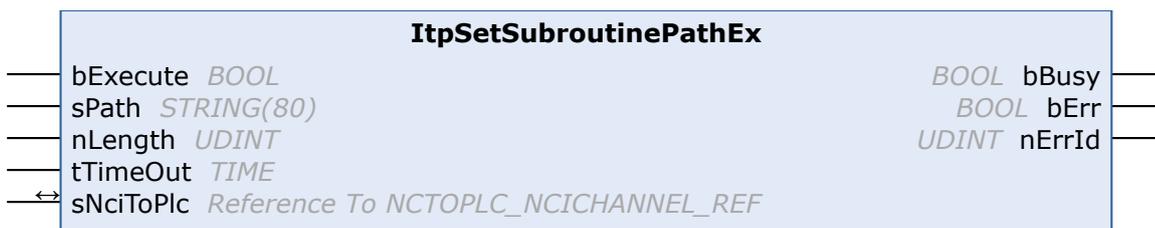
```
VAR
    sPlcToNci AT%Q*: PLCTONC_NCICHANNEL_REF;
    fOverride : LREAL;
END_VAR

fOverride := 47.11;
ItpSetOverridePercent( fOverride, sPlcToNci );
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.42 ItpSetSubroutinePathEx



With `ItpSetSubroutinePathEx` function block, the search path for subroutines can optionally be set.

If a subroutine still has to be integrated, the file is searched in the following order:

1. optional search path (`ItpSetSubroutinePath`)
2. path from which the main program was loaded
3. TwinCAT\Mc\Nci directory

Only one optional path can take effect, which remains active until it is overwritten with another path or an empty string.

After a TwinCAT restart, the path has to be re-assigned.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  sPath         : STRING;
  nLength       : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.
sPath: Optional path for subroutines. Is deactivated with an empty string
nLength: String length
tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |▶ 323|)

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId        : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.
bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.
nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Not Available for GST
 This function block is not available if the GST interpreter is employed.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.43 ItpSetToolDescNullEx



FB ItpSetToolDescNullEx overwrites all tool parameters (incl. number & type) of the channel with zero.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  tTimeOut      : TIME;
END_VAR
```

bExecute: A rising edge results in overwriting of all tool parameters of the NC channel with zero.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy          : BOOL;
  bErr           : BOOL;
  nErrId         : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ItpWriteToolDescEx](#) |> 246]

[ItpReadToolDescEx](#) |> 229]



Not Available for GST

This function block is not available if the GST interpreter is employed.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.44 ItpSetZeroShiftNullEx



The function block ItpSetZeroShiftNullEx overwrites all zero shifts of the channel with zero.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  tTimeOut     : TIME;
END_VAR
```

bExecute: A rising edge results in overwriting of all zero shifts of the NC channel with zero.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Also refer to:

- [ItpWriteZeroShiftEx](#) |> 247].
- [ItpReadZeroShiftEx](#) |> 230].

● Not Available for GST

i This function block is not available if the GST interpreter is employed.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.45 ItpSingleBlock



The ItpSingleBlock function block activates or deactivates single block mode in the NCI. Block relaying can be triggered directly from the PLC with the input 'bTriggerNext'. Alternatively the Start button of the interpreter (F5) can be used in the XAE.

A more detailed description can be found in the [interpreter documentation \[► 129\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecuteModeChange : BOOL;
  nMode               : E_ItpSingleBlockMode;
  bTriggerNext       : BOOL;
  tTimeOut           : TIME;
END_VAR
```

bExecuteModeChange: Single block mode (nMode) is activated through a rising edge at this input.

nMode: Operation mode for single block (cf. single block mode):

- ItpSingleBlockOff: single block off
- ItpSingleBlockNck: single block in NC kernel
- ItpSingleBlockIntp: single block in interpreter

i ItpSingleBlockIntp is not available if the GST interpreter is used.

bTriggerNext: Block relaying is triggered by a rising edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[► 323\]](#))

```
TYPE E_ItpSingleBlockMode:
(
  ItpSingleBlockOff := 0,
  ItpSingleBlockNck := 1,
  ItpSingleBlockIntp := 16#4000
);
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.46 ItpStartStopEx



The function block ItpStartStopEx starts or stops the NC channel.

VAR_INPUT

```
VAR_INPUT
  bStart      : BOOL;
  bStop       : BOOL;
  tTimeOut    : TIME;
END_VAR
```

bStart: A positive edge starts the NC channel

bStop: A positive edge stops the NC channel. A stop command deletes all the tables in the NC and brings the axes to a controlled halt.



The bStop input has a higher priority than the bStart input, so that if both inputs receive a positive edge, a channel stop will be executed.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy          : BOOL;
  bErr           : BOOL;
  nErrId         : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.47 ItpStepOnAfterEStopEx



The function block ItpStepOnAfterEStopEx enables further processing of the parts program after a programmed EStopEx.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    tTimeout      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

tTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy          : BOOL;
    bErr           : BOOL;
    nErrId         : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ItpEStopEx](#) | 207]

[ItplsEStopEx](#) | 224]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.48 ItpWriteRParamsEx



The function block ItpWriteRParamsEx writes R-parameters into the NC.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  pAddr         : DWORD;
  nIndex        : DINT;
  nCount        : DINT;
  tTimeOut      : TIME;
  sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

bExecute: A rising edge starts the write operation.

pAddr: Address of the variables containing the data to be written. Data are used directly from the specified address, i.e. nIndex is not to be interpreted as offset from pAddr. The data are usually read from an array of type LREAL, which has to be defined by the user.

nIndex: Describes the index of the R-parameter to be written from an NC perspective.

nCount: Number of R-parameters to be written

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |▶ 323|)

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Sample

```
VAR
  arrfRParam90to99 : ARRAY[0..9] OF LREAL;
  fbWriteRParam     : ItpWriteRParamsEx;
```

```

n          : INT := 0;
bWriteParam  : BOOL := FALSE;
sNciToPlc AT%I* : NCTOPLC_NCICHANNEL_REF;
END_VAR

FOR n:=0 TO 9 DO
  arrfRParam90to99[n] := 90 + n;
END_FOR

fbWriteRParam(
  bExecute := bWriteParam,
  pAddr := ADR( arrfRParam90to99[0] ),
  nIndex := 90,
  nCount := 10,
  tTimeOut := T#200ms,
  sNciToPlc := sNciToPlc );

```

In this example the parameters R90 to R99 are written from an NC perspective.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.49 ItpWriteToolDescEx



The function block **ItpWriteToolDescEx** writes a block of tool parameters.

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  nDNo         : UDINT;
  tTimeOut     : TIME;
END_VAR

```

bExecute: The command is triggered by a rising edge at this input.

nDNo: D-word for which the tool parameters are to be read. nDNo can have values between 1 and 255.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
  sToolDesc    : ToolDesc;
END_VAR

```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

sToolDesc: The structure that contains the new tool parameters. This structure is only accessed for reading. The meaning of the parameters depends on the tool type, and can be found in the [tool data](#) [▶ 177].

```

TYPE ToolDesc:
STRUCT
  nToolNumber  : UDINT; (*valid range from 0 .. 65535*)
  nToolType    : UDINT;
  fParam       : ARRAY [2..15] OF LREAL;
END_STRUCT
END_TYPE

```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ItpReadToolDescEx \[► 229\]](#)

[ItpSetToolDescNullEx \[► 239\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.50 ItpWriteZeroShiftEx



The function block **ItpWriteZeroShiftEx** writes the shift components X, Y and Z for the specified zero shift.

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  nZsNo       : UDINT;
  tTimeOut    : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nZsNo: Number of the zero shift.

G54 to G59 are zero shifts at the NC. G58 and G59 can only be edited from the NC program. The valid range of values for 'nZsNo' is therefore from 54 to 57.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc AT%I*: NCTOPLC_NCICHANNEL_REF;
  sZeroShiftDesc : ZeroShiftDesc;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC NCICHANNEL REF \[► 323\]](#))

sZeroShiftDesc: The structure containing the components of the zero shift. This structure is only accessed for reading.

```
TYPE ZeroShiftDesc:
STRUCT
  fShiftX      : LREAL;
  fShiftY      : LREAL;
  fShiftZ      : LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).



For reasons of compatibility every zero shift that can be set has two parameters (coarse and fine) for each axis. When using this function block to write a new zero shift, the new value is written into the 'fine parameter'. A value of 0.0 is entered into the 'coarse parameter'.

This makes it possible to use a function block such as [ltpReadZeroShiftEx \[► 230\]](#) to read and modify a zero shift and to send it back to the NC.

See also:

- [ltpReadZeroShiftEx \[► 230\]](#)
- [ltpSetZeroShiftNullEx \[► 240\]](#)

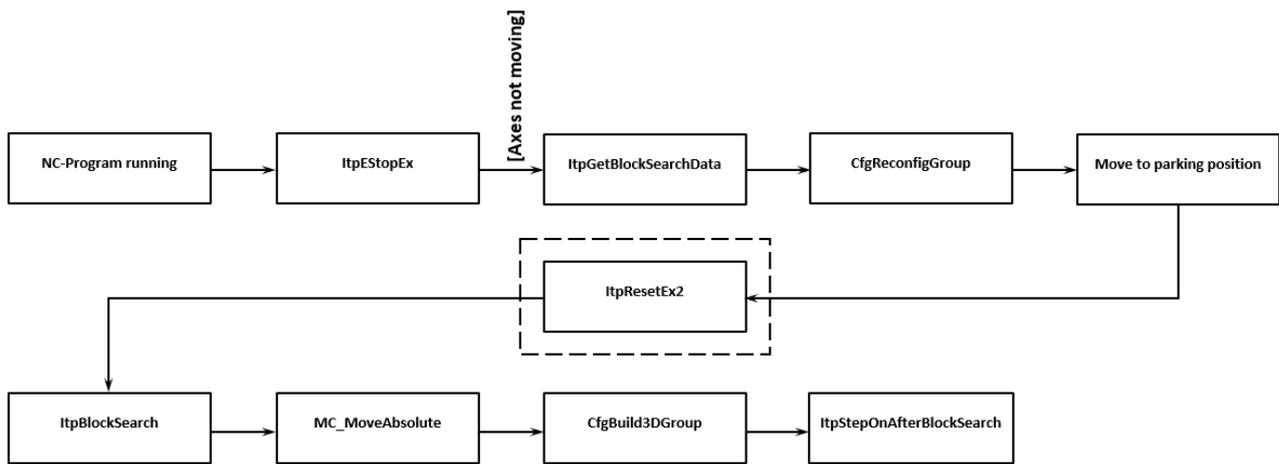
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.51 Blocksearch

Blocksearch can be used to interrupt a program for a tool change or at the end of a shift. After the interruption the program can continue at the previous position.

The diagram illustrates how the block search is used.



6.1.2.51.1 ItpBlocksearch

ItpBlockSearch			
— bExecute	BOOL		— BOOL bBusy
— nBlockId	UDINT		— BOOL bErr
— eBlockSearchMode	E_ItpBlockSearchMode		— UDINT nErrId
— eDryRunMode	E_ItpDryRunMode		— BOOL bDone
— fLength	LREAL	ST_ItpBlockSearchStartPosition	— sStartPosition
— sPrgName	STRING(255)		
— nPrgLength	UDINT		
— tTimeOut	TIME		
— sAxesList	ST_ItpAxes		
— sOptions	ST_ItpBlockSearchOptions		
← sNciToPlc	Reference To NCTOPLC_NCICHANNEL_REF		

The function block ItpBlocksearch sets the interpreter to the point defined at the inputs. If Blocksearch is executed during the first segment that contains a movement, the output sStartPosition of the function block ItpBlocksearch may return wrong values. For this reason, Blocksearch should only be used from the second segment.

The input values can be taken from function block ItpGetBlocksearchData [▶ 252] or set manually. Once the interpreter has been set to the defined location with ItpBlocksearch, the motion can continue with ItpStepOnAfterBlocksearch [▶ 253] at the position indicated at output sStartPosition.

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  nBlockId     : UDINT;
  eBlockSearchMode : E_ItpBlockSearchMode;
  eDryRunMode  : E_ItpDryRunMode;
  fLength      : LREAL;
  sPrgName     : STRING(255);
  nPrgLength   : UDINT;
  tTimeOut    : TIME;
  sAxesList    : ST_ItpAxes;
  sOptions     : ST_ItpBlockSearchOptions;
END_VAR
    
```

bExecute: The command is triggered by a rising edge at this input.

nBlockId: Block number or EntryCounter of the segment in the NC program used as starting point.

eBlockSearchMode: Defines whether the specified nBlockId is a block number (e.g. N4711) or continuous EntryCounter. A prerequisite for using the block number is that it is unique. See ItpBlocksearch [▶ 250].

eDryRunMode: Defines which program lines are executed and which are skipped. See [ItpBlocksearch \[► 251\]](#).

fLength: Entry point within the segment selected with nBlockId in percent.

sPrgName: Name or path of the program to be executed.

nPrgLength: Indicates the length of string sPrgName.

tTimeout: ADS timeout delay

sAxesList: Definition of the axes in the NCI group. See [ItpBlocksearch \[► 251\]](#).

sOptions: Provides information on retrace.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[► 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy              : BOOL;
  bErr               : BOOL;
  nErrId             : UDINT;
  bDone              : BOOL;
  sStartPosition     : ST_ItpBlockSearchStartPosition;
END_VAR
```

bBusy: Remains TRUE until the function block has executed a command request, but no longer than the time specified at the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs.

bErr: Becomes TRUE if an error occurs during command execution. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS Return Codes or in the Overview of NC errors (error codes above 0x4000).

bDone: The output becomes TRUE when the command was executed successfully.

sStartPosition: Indicates the start position from which the NC program continues. The individual axes should be moved to this position before [ItpStepOnAfterBlocksearch \[► 253\]](#) is executed. See [ItpBlocksearch \[► 251\]](#)

E_ItpBlockSearchMode

E_ItpBlockSearchMode is used to define in which way the block search is executed.

```
TYPE E_ItpBlockSearchMode :
(
  ItpBlockSearchMode_Disable      := 0,
  ItpBlockSearchMode_BlockNo     := 1,
  ItpBlockSearchMode_EntryCounter := 2
);
END_TYPE
```

ItpBlockSearchMode_Disable: Block search disabled (initial value).

ItpBlockSearchMode_BlockNo: The block search is executed via the block number (e.g. N4711) programmed by the user in the NC program. A prerequisite is that the user-defined block number is unique.

ItpBlockSearchMode_EntryCounter: The block search is executed via a unique EntryCounter. This EntryCounter is implicitly unique, but it is not visible to the user in the NC program.

E_ItpDryRunMode

The enumeration `E_ItpDryRunMode` enumerates those ways how the programmed blocks from the beginning of the program up to the place searched for shall be handled.

```
TYPE E_ItpDryRunMode :
(
  ItpDryRunMode_Disable           := 0,
  ItpDryRunMode_SkipAll           := 1,
  ItpDryRunMode_SkipMotionOnly    := 2,
  ItpDryRunMode_SkipDwellAndMotion := 3
);
END_TYPE
```

ItpDryRunMode_Disable: DryRun disabled (initial value).

ItpDryRunMode_SkipAll: All previous blocks are skipped. R-parameters are written.

ItpDryRunMode_SkipMotionOnly: Only movement blocks are skipped. R-parameters are written, and dwell times and M-functions are executed.

ItpDryRunMode_SkipDwellAndMotion: Movement blocks and dwell times are skipped. R-parameters are written and M-functions are executed.

ST_ItpAxes

The structure `ST_ItpAxes` contains the axes that were in the NCI group during program execution. The interpolation group should not be built when blocksearch is executed. In order to still have a reference to the group axes, the structure `ST_ItpAxes` must be filled with the group axes.

```
TYPE ST_ItpAxes :
STRUCT
  nAxisIds          : ARRAY[1..8] OF UDINT;
END_STRUCT
END_TYPE
```

nAxisIds: Array of axes that were in the NCI group. The order is `nAxisIds[1]=X`, `nAxisIds[2]=Y`, `nAxisIds[3]=Z`, `nAxisIds[4]=Q1`, `nAxisIds[5]=Q2`... The axis ID can be read from the cyclic axis interface.

St_ItpBlockSearchOptions

The structure contains additional Blocksearch options.

```
TYPE ST_ItpBlockSearchOptions :
STRUCT
  bIsRetrace           : BOOL:= FALSE;
  bRetraceBackward    : BOOL:= FALSE;
  bScanStartPos       : BOOL:= FALSE;
END_STRUCT
END_TYPE
```

bIsRetrace: Indicates whether the retrace functionality is active.

bRetraceBackward: Indicates whether backward movement took place on the path.

bScanStartPos: `bScanStartPos`: Specifies whether or not the current axis positions should be read at the start of the program. In combination with `ST_ItpAxesList`, please set this input to `TRUE`. Setting this input to `FALSE` only makes sense for old projects (compatibility reasons).

ST_ItpBlockSearchStartPosition

The structure indicates the position at which the NC program continues after a block search. The user is responsible for moving the axes to the corresponding positions.

```
TYPE ST_ItpBlockSearchStartPosition :
STRUCT
  sStartPosition      : ARRAY[1..8] OF LREAL;
END_STRUCT
END_TYPE
```

sStartPosition: Array of axis positions at which the NC program continues.

The order is sStartPosition[1]=X, sStartPosition [2]=Y, sStartPosition [3]=Z, sStartPosition [4]=Q1, sStartPosition [5]=Q2...

Voraussetzungen

Entwicklungsumgebung	Zielformat	Einzubindende SPS Bibliotheken
Classic Dialect Interpreter: TwinCAT V3.1.0 GST Interpreter: TwinCAT V3.1.4024.20	PC oder CX (x86 oder x64)	Tc2_NCI

6.1.2.51.2 ItpGetBlocksearchData



The function block ItpGetBlocksearchData reads the current position on the path. Usually this command is called at standstill. Subsequently [ItpBlockSearch \[► 249\]](#) can be used to set the interpreter to the position stored in sBlockSearchData.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  tTimeout      : TIME;
END_VAR
```

bExecute: The command is triggered by a positive edge at this input.

tTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[► 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy          : BOOL;
  bErr           : BOOL;
  nErrId         : UDINT;
  sBlockSearchData : ST_ItpBlockSearchData;
END_VAR
```

bBusy: Remains TRUE until the function block has executed a command request, but no longer than the time specified at the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs.

bErr: Becomes TRUE if an error occurs during command execution. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

sBlockSearchData: Contains information on the current position on the path.

```
TYPE ST_ItpBlockSearchData :
STRUCT
  fLength        : LREAL; (* remaining distance of actual movement block in percent*)
  nBlockNo       : UDINT; (* number of the actual block *)
END_STRUCT
```

```
nBlockCounter      : UDINT; (* counter value of the actual block *)
bIsRetrace         : BOOL; (* indicates whether Retrace is active*)
bRetraceBackward  : BOOL; (* indicates whether backward movement took place on the path*)
END_STRUCT
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Classic Dialect Interpreter: TwinCAT V3.1.0 GST Interpreter: TwinCAT V3.1.4024.20	PC oder CX (x86 oder x64)	Tc2_NCI

6.1.2.51.3 ItpStepOnAfterBlocksearch



Starts the motion after a block search.

The axes first have to be moved to the positions output by [ItpBlocksearch](#) [▶ 249].

VAR_INPUT

```
VAR_INPUT
  bExecute          : BOOL;
  tTimeout          : TIME;
END_VAR
```

bExecute: The command is triggered by a positive edge at this input.

bTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc        : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy            : BOOL;
  bErr             : BOOL;
  nErrId           : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
Classic Dialect Interpreter: TwinCAT V3.1.0 GST Interpreter: TwinCAT V3.1.4024.20	PC oder CX (x86 oder x64)	Tc2_NCI

6.1.2.52 Retrace

6.1.2.52.1 ItpEnableFeederBackup



The function block `ItpEnableFeederBackup` enables storing of the path for retracing. It has to be activated once before the NC program (G-Code) is started. If the [Blocksearch \[▶ 248\]](#) functionality is used, `ItpEnableFeederBackup` has to be activated before `ItpBlocksearch [▶ 249]` is called. Feeder backup is executed as long as a TwinCAT restart or `bEnable = FALSE` is triggered with a rising edge at `bExecute`.

If feeder backup is not enabled, retracing does not work. This can be verified via [ItpsFeederBackupEnabled \[▶ 255\]](#).

VAR_INPUT

```
VAR_INPUT
    bEnable      : BOOL;
    bExecute     : BOOL;
    tTimeOut     : TIME;
END_VAR
```

bEnable: TRUE: enables feeder backup, FALSE: disables feeder backup

bExecute: The command is triggered by a positive edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc   : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF \[▶ 323\]](#))

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bErr        : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

● Not available for GST

i This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.52.2 ItpIsFeederBackupEnabled



The function block ItpIsFeederBackupEnabled indicates whether feeder backup is enabled. Feeder backup must be enabled before reversing can take place. This activates storing of the path.

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  tTimeOut     : TIME;
END_VAR
  
```

bExecute: The command is triggered by a positive edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
  
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy         : BOOL;
  bEnabled      : BOOL;
  bErr          : BOOL;
  nErrId       : UDINT;
END_VAR
  
```

bBusy: The bBusy output remains TRUE until the function block has executed a command, with the maximum duration specified by the time associated with the 'Timeout' input. While bBusy = TRUE, no new instruction will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bEnabled: TRUE: Backup list for tracing is enabled, FALSE: Backup list for tracing is disabled

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. If the function block has a timeout error, 'Error' is TRUE and 'nErrId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

● Not available for GST

i This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.52.3 ItpIsFeedFromBackupList



The function ItpIsFeedFromBackupList becomes TRUE when the feed entries (SAF & SVB) were sent from the backup list. During backward movement all entries are sent from the backup list. If the program is executed in forward mode, the first entries usually also originate from the backup list. This is dependent of the number of retraced entries and the number of entries in the SVB and SAF tables at the time at which tracing was called. All further commands originate from the ‚original‘ code.

While the NCI is processing the backup list, not all functions are available or meaningful. Here are a few examples:

- Decoder stops such as @714 are not evaluated
- Modifications of R-parameters do not take effect as long as the motion takes place on the backup path (forward or backward). R-parameters modifications take effect again as soon as the path data no longer come from the backup list.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) | 323])

● Not available for GST

i This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.52.4 ItpIsFirstSegmentReached



ItpIsFirstSegmentReached is a function that determines whether the program start position is reached during retracing, based on the cyclic channel interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |▶ 323|)

Return value

The function returns TRUE when the start position of the G-Code program is reached. If the version number of the cyclic channel interface is less than 6, the return value is always FALSE.

● Not available for GST
i This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.52.5 ItpIsMovingBackwards



ItpIsMovingBackwards is a function that determines whether backward movement takes place on the path of the current G-Code program, based on the cyclic channel interface.

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |▶ 323|)

Return value

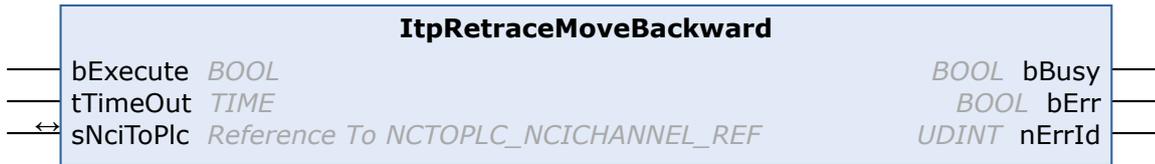
The function returns TRUE when backward movement takes place on the path. If the version number of the cyclic channel interface is less than 6, the return value is always FALSE.

● Not available for GST
i This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.52.6 ItpRetraceMoveBackward



The function block ItpRetraceMoveBackward deals with the geometric entries at the actual position at the start of the part program (G-Code).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a positive edge at this input.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sNciToPlc    : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) |> 323])

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Procedure

1. Activate feeder backup list (see [ItpEnableFeederBackup](#) |> 254)
 - ⇒ The NC program is stopped with [ItpEStopEx](#) |> 207]
2. Wait and ensure that all axes in the group are at standstill
3. Call ItpRetraceMoveBackward
4. Stop backward movement with ItpEStop, otherwise the program returns to the start
5. Call [ItpRetraceMoveForward](#) |> 259] to move forward again
6. Call ItpEStopEx and ItpRetraceMoveBackward etc., if required.

Note Do not use in conjunction with vertex blending. M-functions are suppressed during backward movement.



Not available for GST

This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.2.52.7 ItpRetraceMoveForward



The function block `ItpRetraceMoveForward` transfers all entries from the current block (e.g. position) in forward travel direction to the NC kernel. It is called to reverse the direction after `ItpRetraceMoveBackward` [▶ 258] was called.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    tTimeout      : TIME;
END_VAR
```

bExecute: The command is triggered by a positive edge at this input.

tTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sNciToPlc      : NCTOPLC_NCICHANNEL_REF;
END_VAR
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: `NCTOPLC_NCICHANNEL_REF` [▶ 323])

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy         : BOOL;
    bErr          : BOOL;
    nErrId        : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also: `ItpRetraceMoveBackward` [▶ 258]



Not available for GST

This function block is not available if the GST interpreter is used.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

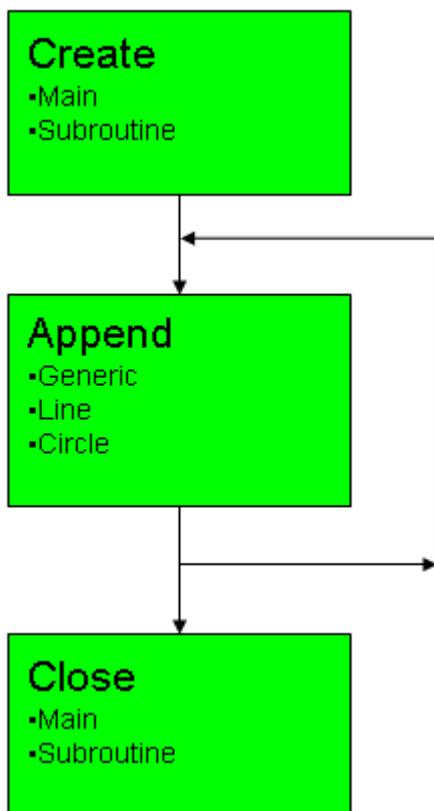
6.1.3 Parts program generator

The function blocks `ItpPpg*` provide an option for creating a parts program (G-Code file) from the PLC. During program generation a distinction is made between a main program (`ItpPpgCreateMain` [▶ 266]) and a subroutine (`ItpPpgCreateSubroutine` [▶ 266]).

Subsequently `ItpPpgAppend*` can be used to add various NC lines. The following function blocks are available:

- `ItpPpgAppendGeoLine` [▶ 263] adds a linear motion.
- `ItpPpgAppendGeoCircleByRadius` [▶ 262] adds a circle with radius specification.
- `ItpPpgAppendGenericBlock` [▶ 261] inserts a self-defined line, such as activation of rounding or M-functions.

Once the parts program is complete, it is closed with the routines `ItpPpgCloseMain` [▶ 264] or `ItpPpgCloseSubroutine` [▶ 265].



The following function blocks can be used:

Function Block	Description
ItpPpgAppendGenericBlock [▶ 261]	Appends a generic NC line to a specified parts program
ItpPpgAppendGeoCircleByRadius [▶ 262]	Adds a circle to a specified parts program
ItpPpgAppendGeoLine [▶ 263]	Adds a linear motion to a specified parts program
ItpPpgCloseMain [▶ 264]	Closes a previously opened parts program
ItpPpgCloseSubroutine [▶ 265]	Closes a previously opened subroutine
ItpPpgCreateMain [▶ 266]	Opens or generates a parts program
ItpPpgCreateSubroutine [▶ 266]	Opens or generates a subroutine

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.1 ItpPpgAppendGenericBlock



The function block `ItpPpgAppendGenericBlock` adds a generic line to the parts program. It can be used to activate an M-function or rounding, for example.

Before the actual call, call [ItpPpgCreateMain \[▶ 266\]](#) or [ItpPpgCreateSubroutine \[▶ 266\]](#).

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  sPathName     : STRING;
  sBlock        : STRING;
  tTimeOut      : TIME;
END_VAR
    
```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the parts program including path name

sBlock: Generic line to be added to the parts program

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId        : UDINT;
END_VAR
    
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

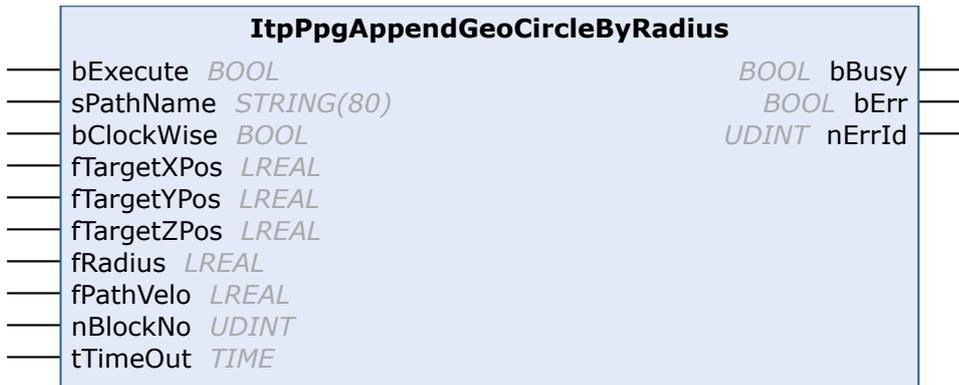
bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.2 ItpPpgAppendGeoCircleByRadius



The function block `ItpPpgAppendGeoCircleByRadius` adds a circular motion to the parts program. The circle is parameterized by the radius.

Before the actual call, call [ItpPpgCreateMain](#) [▶ 266] or [ItpPpgCreateSubroutine](#) [▶ 266].

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  sPathName     : STRING;
  bClockWise    : BOOL;
  fTargetXPos   : LREAL;
  fTargetYPos   : LREAL;
  fTargetZPos   : LREAL;
  fRadius       : LREAL;
  fPathVelo     : LREAL;
  nBlockNo      : UDINT;
  tTimeOut      : TIME;
END_VAR

```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the parts program including path name

bClockwise: If TRUE, the movement along the circle is clockwise, otherwise counter-clockwise

fTargetXPos: Target position of the X axis

fTargetYPos: Target position of the Y axis

fTargetZPos: Target position of the Z axis

fRadius: Circle radius

fPathVelo: Path velocity

nBlockNo: Line number in the parts program

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.3 ItpPpgAppendGeoLine



The function block ItpPpgAppendGeoLine adds a linear motion to the parts program. In addition to the actual target position, the path velocity and the line number are transferred.

Before the actual call, call [ItpPpgCreateMain \[▶ 266\]](#) or [ItpPpgCreateSubroutine \[▶ 266\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  sPathName     : STRING;
  fTargetXPos   : LREAL;
  fTargetYPos   : LREAL;
  fTargetZPos   : LREAL;
  fPathVelo     : LREAL;
  nBlockNo      : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the parts program including path name

fTargetXPos: Target position of the X axis

fTargetYPos: Target position of the Y axis

fTargetZPos: Target position of the Z axis

fPathVelo: Path velocity

nBlockNo: Line number in the parts program

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

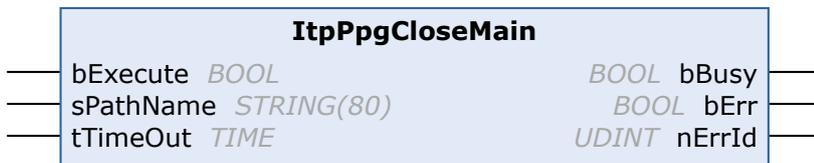
bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.4 ItpPpgCloseMain



The function block ItpPpgCloseMain completes the main program with the corresponding code for the interpreter (M02).

Before the actual call, call [ItpPpgCreateMain](#) [▶ 266].

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  sPathName   : STRING;
  tTimeOut    : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the parts program including path name

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.5 ItpPpgCloseSubroutine



The function block ItpPpgCloseSubroutine completes the subroutine with the corresponding code for the interpreter (M17).

Before the actual call, call [ItpPpgCreateSubroutine](#) [► 266].

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    sPathName     : STRING;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the parts program including path name

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.6 ItpPpgCreateMain



The function block ItpPpgCreateMain generates a new file, which can later be processed as main program. If the file does not yet exist, it is created, otherwise it is overwritten.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    sPathName     : STRING;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the parts program including path name

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.3.7 ItpPpgCreateSubroutine



The function block ItpPpgCreateSubroutine generates a new file, which can later be processed as subroutine. If the file does not yet exist, it is created, otherwise it is overwritten.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  sPathName     : STRING;
  nSubroutineId : UDINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

sPathName: Name of the subroutine including path name

nSubroutineId: Number of the subroutine

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4 Blocks for compatibility with existing programs

i **Function blocks for compatibility**

The purpose of the function blocks listed is to ensure compatibility with existing projects. It is **not** advisable to use these function blocks for new projects. Instead, the equivalent function blocks shown in the table above should be used.

Function Block	Description
ltpDelDtg [▶ 268]	Triggers "Delete Distance to go" in the NC
ltpEStop [▶ 269]	Triggers the NCI EStop
ltpGetBottleNeckLookAhead [▶ 270]	Provides the value of the look-ahead for bottleneck detection
ltpGetBottleNeckMode [▶ 271]	Provides the response mode for bottleneck detection
ltpGetGeoInfoAndHParam [▶ 272]	Reads information of the currently active segment and past and future segments.
ltpGoAhead [▶ 273]	Triggers the GoAhead function
ltpIsEStop [▶ 273]	Determines whether an EStop is executed or pending
ltpLoadProg [▶ 275]	Loads an NC program using program names
ltpReadRParams [▶ 276]	Reads calculation parameters

Function Block	Description
ItpReadToolDesc [▶ 277]	Reads the tool description from the NC
ItpReadZeroShift [▶ 278]	Reads the zero shift from the NC
ItpReset [▶ 279]	Carries out a reset of the interpreter or of the NC channel
ItpResetEx [▶ 280]	Carries out a reset of the interpreter or of the NC channel.
ItpResetFastMFunc [▶ 281]	Resets a fast signal bit
ItpSetBottleNeckLookAhead [▶ 282]	Sets the value of the look-ahead for bottleneck detection
ItpSetBottleNeckMode [▶ 283]	Sets the response mode when bottleneck detection is switched on
ItpSetSubroutinePath [▶ 284]	Optionally sets the search path for subroutines
ItpSetToolDescNull [▶ 285]	Sets all tool parameters (including number and type) to zero
ItpSetZeroShiftNull [▶ 286]	Sets all origins to zero
ItpStartStop [▶ 287]	Starts or stops the interpreter (NC channel)
ItpStepOnAfterEStop [▶ 288]	Enables further processing of the parts program after an NCI EStop
ItpWriteRParams [▶ 289]	Writes calculation parameters
ItpWriteToolDesc [▶ 290]	Writes the tool description into the NC
ItpWriteZeroShift [▶ 291]	Writes the zero shift into the NC

6.1.4.1 ItpDelDtg



The ItpDelDtg function block triggers deletion of the remaining travel. There is a more detailed description in the [Interpreter \[▶ 156\]](#) documentation.

i **Outdated version**

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpDelDtgEx \[▶ 205\]](#).

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    nChnId       : UDINT;
    tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bErr        : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.2 ItpEStop



The function block ItpEStop triggers the NCI EStop and enables a controlled stop on the path. The limit values for the deceleration and the jerk are transferred as parameters. If these are smaller than the currently active dynamic parameters, the transferred parameters are rejected.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpEStopEx \[► 207\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGrpId        : UDINT;
  fDec          : LREAL;
  fJerk         : LREAL;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGrpId: group ID

fDec: Max. deceleration during stopping. If fDec is smaller than the currently active deceleration, fDec is rejected. This ensures that the deceleration occurs with the standard ramp as a minimum.

fJerk: Max. jerk during stopping. If fJerk is smaller than the currently active jerk, fJerk is rejected.

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ItpStepOnAfterEStop](#) [▶ 288]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.3 ItpGetBottleNeckLookAhead



The function block ItpGetBottleNeckLookAhead determines the maximum size of the look-ahead for the bottleneck detection (contour collision monitoring).

There is a more detailed description in the [Interpreter](#) [▶ 189] documentation.

Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpGetBottleNeckLookAheadEx](#) [▶ 209].

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    nChnId        : UDINT;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy         : BOOL;
    bErr          : BOOL;
    nErrId        : UDINT;
    nLookAhead    : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. If the function block has a timeout error, 'Error' is TRUE and 'nErrId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

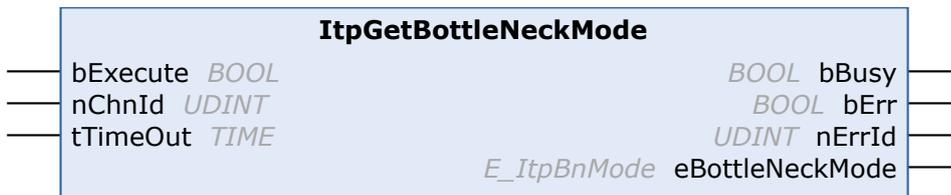
nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

nLookAhead: Value of the look-ahead for bottleneck detection

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.4 ItpGetBottleNeckMode



The function block ItpGetBottleNeckMode reads the behavior in the event of a contour collision (bottleneck).

There is a more detailed description in the [Interpreter \[► 189\]](#) documentation.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpGetBottleNeckModeEx \[► 210\]](#).

VAR_INPUT

```

VAR_INPUT
  bExecute      : BOOL;
  nChnId        : UDINT;
  tTimeout      : TIME;
END_VAR
  
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

tTimeout: ADS Timeout-Delay

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId        : UDINT;
  eBottleNeckMode: E_ItpBnMode
END_VAR
  
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. If the function block has a timeout error, 'Error' is TRUE and 'nErrId' is 1861 (hexadecimal 0x745). Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

eBottleNeckMode: Enum for the behavior in the event of a contour collision

```

TYPE E_ItpBnMode:
(
  ItpBnm_Abort := 0,
  ItpBnm_Adjust := 1,
  ItpBnm_Leave := 2
);
END_TYPE
    
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.5 ItpGetGeoInfoAndHParam



The function block ItpGetGeoInfoAndHParam reads information of the currently active segment and past and future segments. These include block number, H-parameter and residual path length on the segment.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpGetGeoInfoAndHParamEx](#) [▶ 215].

VAR_IN_OUT

```

VAR_IN_OUT
  sNciToPlc          : NCTOPLC_NCICHANNEL_REF;
END_VAR
    
```

sNciToPlc: The structure of the cyclic channel interface from the NCI to the PLC. This structure is only accessed for reading. (type: [NCTOPLC_NCICHANNEL_REF](#) [▶ 323])

VAR_OUTPUT

```

VAR_OUTPUT
  stTab              : ST_ItpPreViewTabEx;
  nErrId             : UDINT;
END_VAR
    
```

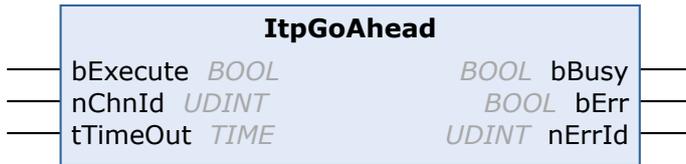
stTab: Structure containing the segment data.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.6 ItpGoAhead



The function block ItpGoAhead may only be used in association with the decoder stop '@717' [▶ 166]. There is a more detailed description of this decoder stop in the [interpreter documentation](#) [▶ 122].

Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpGoAheadEx](#) [▶ 222].

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    nChnId        : UDINT;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

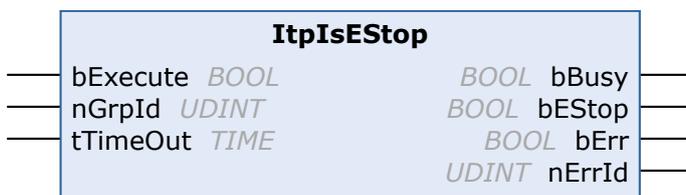
bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.7 ItpIsEStop



Via bEStop, the function block ltpIsEStop provides information as to whether an EStop command was triggered. If bEStop is TRUE, then an EStop was initiated (e.g. ltpEStop). The flag does **not** provide information as to whether the axes have already stopped or are still on the braking ramp.

After the execution of ltpStepOnAfterEStop, ltpIsEStop will once again return FALSE.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ltpIsEStopEx \[► 224\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGrpId       : UDINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGrpId: group ID

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bEStop      : BOOL;
  bErr        : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bEStop: TRUE: EStop command was executed, FALSE: No EStop present

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ltpEStop \[► 269\]](#)

[ltpStepOnAfterEStop \[► 288\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.8 ItpLoadProg



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpLoadProgEx \[► 225\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nChnId        : UDINT;
  sPrg          : STRING;
  nLength       : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: A rising edge at this input triggers execution of the NC program

nChnId: Channel ID

sPrg: Name of the NC program that is executed

nLength: String length of the program name

tTimeOut: ADS Timeout-Delay

Note The NC program is looked up in directory "TwinCAT\Mc\Nci", if no further information is available. It is however also possible to give an absolute path.

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.9 ItpReadRParams



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpReadRParamsEx](#) [▶ 228].

The ItpReadRParams function block reads the NC's calculation parameters, also known as R-parameters. A more detailed description of the calculation parameters can be found [here](#) [▶ 130]. A total of 1000 R-parameters are available, of which the first 900 (0..899) are local, so that they are only visible in the current NC channel. The other 100 (900..999) R-parameters are global, and are thus visible from anywhere in the NC.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    nChnId        : UDINT;
    pAddr         : PVOID;
    nIndex        : DINT;
    nCount        : DINT;
    tTimeOut      : TIME;
END_VAR
```

bExecute: A rising edge starts the read operation

nChnId: ID of the NC channel whose R-parameters are to be read

pAddr: Address of the target variables of the data to be read. The data are written by the NC directly from the specified address. i.e. nIndex is not to be interpreted as offset from pAddr. The data are usually in an array of type LREAL, which has to be defined by the user.

nIndex: Describes the index of the R-parameter to be read from an NC perspective.

nCount: Number of R-parameters to be read

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy        : BOOL;
    bErr         : BOOL;
    nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

[ItpWriteRParams](#) [▶ 289]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.10 ItpReadToolDesc



The ItpReadToolDesc function block reads the tool parameters for the supplied D-word.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpReadToolDescEx](#) [▶ 229].

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nChnId        : UDINT;
  nDNo          : UDINT;
  tTimeout      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

nDNo: D-word for which the tool parameters are to be read. nDNo can have values between 1 and 255.

tTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sToolDesc     : ToolDesc;
END_VAR
```

sToolDesc: A structure into which the tool parameters of nDNo are written. The meaning of the parameters depends on the tool type, and can be found in the [tool data](#) [▶ 177].

```
TYPE ToolDesc:
STRUCT
  nToolNumber   : UDINT; (*valid range from 0 .. 65535*)
  nToolType     : UDINT;
  fParam        : ARRAY [2..15] OF LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy         : BOOL;
  bErr          : BOOL;
  nErrId        : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

see also:

[ItpWriteToolDesc \[▶ 290\]](#); [ItpSetToolDescNull \[▶ 285\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.11 ItpReadZeroShift



The **ItpReadZeroShift** function block reads the offset shift components X, Y and Z for the given zero shift.

Note For reasons of compatibility, there are two entries (coarse and fine) for each axis in each zero shift (e.g. G54). These two entries must be added together. This function block evaluates both the entries and adds them together automatically.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpReadZeroShiftEx \[▶ 230\]](#).

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    nChnId        : UDINT;
    nZsNo         : UDINT;
    tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

nZsNo: Number of the zero shift; on the NC side G54 to G59 are zero shifts. The valid range of values for 'nZsNo' is therefore from 54 to 59.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
    sZeroShiftDesc : ZeroShiftDesc;
END_VAR
```

sZeroShiftDesc: The structure containing the components of the zero shift.

```
TYPE ZeroShiftDesc:
STRUCT
  fShiftX      : LREAL;
  fShiftY      : LREAL;
  fShiftZ      : LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

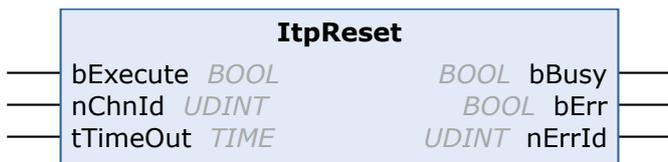
see also:

[ltpWriteZeroShift \[▶ 291\]](#); [ltpSetZeroShiftNull \[▶ 286\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.12 ItpReset



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpResetEx2 \[▶ 231\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nChnId        : UDINT;
  tTimeOut      : TIME;
END_VAR
```

bExecute: A rising edge at this input triggers a reset of the NC channel

nChnId: Channel ID

tTimeOut: ADS Timeout-Delay

Note A reset deletes all tables in the NC. The axes are halted immediately. For this reason a reset should only be carried out either in the event of an error or when the axes are stationary.

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR

```

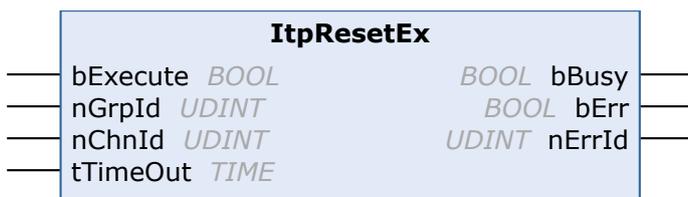
bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.13 ItpResetEx

The function block 'ItpResetEx' executes a channel reset, which deletes all existing tables of the NC channel. In contrast to the conventional [ItpReset](#) [[▶ 279](#)], an active channel is stopped first, before the reset is executed. This simplifies programming in the PLC, since no explicit check is necessary to ascertain whether the axes are still in motion.

**Outdated version**

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpResetEx2](#) [[▶ 231](#)].

VAR_INPUT

```

VAR_INPUT
  bExecute    : BOOL;
  nGrpId      : UDINT;
  nChnId      : UDINT;
  tTimeOut    : TIME;
END_VAR

```

bExecute: The command is triggered by a rising edge at this input.

nGrpId: group ID

nChnId: Channel ID

tTimeOut: ADS timeout delay (the bBusy signal can be active for longer than tTimeOut)

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

see also: [ItpStartStop](#) [▶ 287]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.14 ItpResetFastMFunc



This function block represents an alternative to Auto-reset or reset with another M-function (reset list during parameterization of the M-function). For the sake of clarity, mixed operation involving resetting with an M-function and this function block should be avoided.

The fast M-function [▶ 161] **nMFuncNo** is reset with a rising edge at input **bExecute**. In the event of the M-function not being available, **no** error is returned.

Outdated version

i The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpResetFastMFuncEx](#) [▶ 232].

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  nChnId      : UDINT;
  nMFuncNo    : UINT;
  tTimeOut    : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

nMFuncNo: Flying M-function that is to be reset

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR

```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.15 ItpSetBottleNeckLookAhead

The function block ItpSetBottleNeckLookAhead determines the maximum number of segments the system may look ahead for bottleneck detection (contour collision monitoring). Note that segments, which were added as a result of radius compensation (e.g. additional segments at acute angles) are taken into account.

There is a more detailed description in the [Interpreter \[► 189\]](#) documentation.

i Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpSetBottleNeckLookAheadEx \[► 233\]](#).

VAR_INPUT

```

VAR_INPUT
  bExecute    : BOOL;
  nChnId      : UDINT;
  nLookAhead  : UDINT;
  tTimeOut    : TIME;
END_VAR

```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

nLookAhead: Specifies the look-ahead value

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.16 ItpSetBottleNeckMode



The function block ItpSetBottleNeckMode specifies the behavior in the event of a contour collision (bottleneck).

There is a more detailed description in the [Interpreter \[▶ 189\]](#) documentation.

Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpSetBottleNeckModeEx \[▶ 234\]](#).

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nChnId        : UDINT;
  eBottleNeckMode: E_ItpBnMode;
  tTimeOut      : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

eBottleNeckMode: Enum for the behavior in the event of a contour collision

tTimeOut: ADS Timeout-Delay

```
TYPE E_ItpBnMode:
(
  ItpBnm_Abort   := 0,
  ItpBnm_Adjust := 1,
  ItpBnm_Leave    := 2
);
END_TYPE
```

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR

```

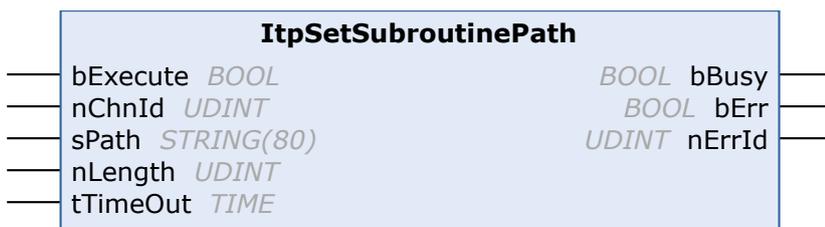
bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.17 ItpSetSubroutinePath

With ItpSetSubroutinePath function block, the search path for subroutines can optionally be set.

If a subroutine still has to be integrated, the file is searched in the following order:

- optional search path (ItpSetSubroutinePath)
- path from which the main program was loaded
- TwinCAT\Mc\Nci directory

Only one optional path can be active at any one time. It remains active until it is

- overwritten with another path or
- with an empty string

After a TwinCAT restart, the path has to be re-assigned.

Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpSetSubroutinePathEx \[► 238\]](#).

Interface

```

VAR_INPUT
  bExecute    : BOOL;
  nChnId      : UDINT;
  sPath       : STRING;
  nLength     : UDINT;
  tTimeOut    : TIME;
END_VAR

```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

sPath: Optional path for subroutines; is disabled with an empty string.

nLength: String length

tTimeout: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.18 ItpSetToolDescNull



FB ItpSetToolDescNull overwrites all tool parameters (incl. number & type) of the channel with zero.

Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpSetToolDescNullEx](#) [▶ 239].

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  nChnId      : UDINT;
  tTimeout    : TIME;
END_VAR
```

bExecute: A rising edge results in overwriting of all tool parameters of the NC channel with zero.

nChnId: ID of the NC channel

tTimeout: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

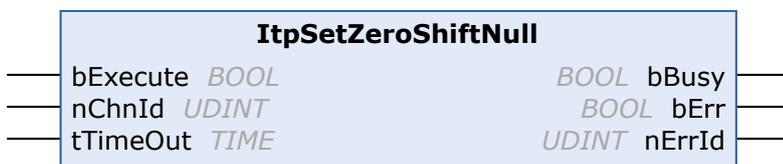
See also:

- [ItpWriteToolDesc \[▶ 290\]](#),
- [ItpReadToolDesc \[▶ 277\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.19 ItpSetZeroShiftNull



FB ItpSetZeroShiftNull overwrites all zero shifts of the channel with zero.

Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block ItpSetZeroShiftNullEx.

VAR_INPUT

```
VAR_INPUT
  bExecute    : BOOL;
  nChnId     : UDINT;
  tTimeOut   : TIME;
END_VAR
```

bExecute: A rising edge results in overwriting of all zero shifts of the NC channel with zero.

nChnId: ID of the NC channel

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

- [ItpWriteZeroShift \[► 291\]](#)
- [ItpReadZeroShift \[► 278\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.20 ItpStartStop



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpStartStopEx \[► 243\]](#).

Interface

```

VAR_INPUT
  bStart      : BOOL;
  bStop       : BOOL;
  nChnId      : UDINT;
  tTimeOut    : TIME;
END_VAR
    
```

bStart: A positive edge starts the NC channel

bStop: A positive edge stops the NC channel. A stop command deletes all the tables in the NC and brings the axes to a controlled halt.

nChnId: Channel ID

tTimeOut: ADS Timeout-Delay

NOTE! The **bStop** input has a higher priority than the **bStart** input, so that if both inputs receive a positive edge, a channel stop will be executed.

VAR_OUTPUT

```

VAR_OUTPUT
  bBusy       : BOOL;
  bErr        : BOOL;
  nErrId      : UDINT;
END_VAR
    
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.21 ItpStepOnAfterEStop



The function block ItpStepOnAfterEStop enables further processing of the parts program after a programmed EStop.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpStepOnAfterEStopEx](#) [▶ 244].

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nGrpId       : UDINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nGrpId: group ID

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy        : BOOL;
  bErr         : BOOL;
  nErrId       : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

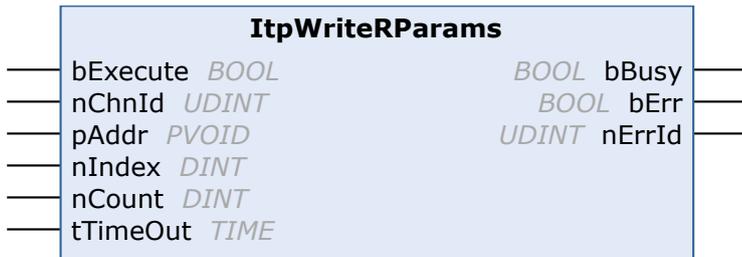
See also:

- [ItpEStop](#) [[▶ 269](#)]
- [ItplsEStop](#) [[▶ 273](#)]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.22 ItpWriteRParams



The ItpWriteRParams function block writes R-parameters into the NC.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpWriteRParamsEx](#) [[▶ 245](#)].

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nChnId       : UDINT;
  pAddr        : PVOID;
  nIndex       : DINT;
  nCount       : DINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: A rising edge starts the write operation.

nChnId: ID of the NC channel whose R-parameters are to be written.

pAddr: Address of the variables containing the data to be written. Data are used directly from the specified address, i.e. nIndex is not to be interpreted as offset from pAddr. The data are usually in an array of type LREAL, which has to be defined by the user.

nIndex: Describes the index of the R-parameter to be written from an NC perspective.

nCount: Number of R-parameters to be written

tTimeOut: ADS Timeout-Delay

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Sample

In this example the parameters R90 to R99 are written from an NC perspective.

```

VAR
  arrfRParam90to99 : ARRAY[0..9] OF LREAL;
  fbWriteRParam    : ItpWriteRParams;
  n                : INT := 0;
  bWriteParam      : BOOL := FALSE;
END_VAR

FOR n:=0 TO 9 DO
  arrfRParam90to99[n] := 90 + n;
END_FOR

fbWriteRParam(
  bExecute := bWriteParam,
  nChnId   := 2,
  pAddr    := ADR( arrfRParam90to99[0] ),
  nIndex   := 90,
  nCount   := 10,
  tTimeout := T#200ms );
    
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.23 ItpWriteToolDesc



The **ItpWriteToolDesc** function block writes a block of tool parameters.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ItpWriteToolDescEx](#) [▶ 246].

VAR_INPUT

```

VAR_INPUT
  bExecute : BOOL;
  nChnId   : UDINT;
  nDNo     : UDINT;
  tTimeout : TIME;
END_VAR
    
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

nDNo: D-word for which the tool parameters are to be read. nDoNo can have values between 1 and 255.

tTimeout: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sToolDesc      : ToolDesc;
END_VAR
```

sToolDesc: The structure that contains the new tool parameters. This structure is only accessed for reading. The meaning of the parameters depends on the tool type, and can be found in the [tool data \[► 177\]](#).

```
TYPE ToolDesc:
STRUCT
  nToolNumber    : UDINT; (*valid range from 0 .. 65535*)
  nToolType      : UDINT;
  fParam        : ARRAY [2..15] OF LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy          : BOOL;
  bErr           : BOOL;
  nErrId         : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

- [ItpReadToolDesc \[► 277\]](#)
- [ItpSetToolDescNull \[► 285\]](#)

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.4.24 ItpWriteZeroShift



The function block **ItpWriteZeroShift** writes the shift components X, Y and Z for the specified zero shift.

For reasons of compatibility every zero shift that can be set has two parameters (coarse and fine) for each axis. When using this function block to write a new zero shift, the new value is written into the 'fine parameter'. A value of 0.0 is entered into the 'coarse parameter'. This makes it possible to use a function block such as [ItpReadZeroShift \[► 278\]](#) to read and modify a zero shift and to send it back to the NC.



Outdated version

The sole purpose of the function block is to ensure compatibility with existing projects. For new projects please use the function block [ltpWriteZeroShiftEx](#) [[▶ 247](#)].

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  nChnId       : UDINT;
  nZsNo        : UDINT;
  tTimeOut     : TIME;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

nChnId: Channel ID

nZsNo: Number of the zero shift.

On the NC side G54 to G59 are zero shifts; G58 and G59 can only be edited from the NC program. The valid range of values for 'nZsNo' is therefore from 54 to 57.

tTimeOut: ADS Timeout-Delay

VAR_IN_OUT

```
VAR_IN_OUT
  sZeroShiftDesc : ZeroShiftDesc;
END_VAR
```

sZeroShiftDesc: The structure containing the components of the zero shift. This structure is only accessed for reading.

```
TYPE ZeroShiftDesc:
STRUCT
  fShiftX : LREAL;
  fShiftY : LREAL;
  fShiftZ : LREAL;
END_STRUCT
END_TYPE
```

VAR_OUTPUT

```
VAR_OUTPUT
  bBusy      : BOOL;
  bErr       : BOOL;
  nErrId     : UDINT;
END_VAR
```

bBusy: This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.

bErr: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in 'nErrId'. Is reset to FALSE by the execution of a command at the inputs.

nErrId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

See also:

- [ltpReadZeroShift](#) [[▶ 278](#)]
- [ltpSetZeroShiftNull](#) [[▶ 291](#)]

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.5 Obsolete

6.1.5.1 F_GetVersionTcNciUtilities



This function returns part of the three-part version number of the TwinCAT 2 PLC library TcNciUtilities.lib as UINT.

Outdated version

The sole purpose of this function is to ensure compatibility with existing projects. For new projects please use the global structure stLibVersion_Tc2_NCI.

VAR_INPUT

```
FUNCTION F_GetVersionNciUtilities
VAR_INPUT
    nVersionElement      : INT;
END_VAR
```

nVersionElement: Part of the version number to be read (range: [1..3])

Return value

F_GetVersionNciUtilities: Version number

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.5.2 Get_TcNcCfg_Version



This function returns the version number of the TwinCAT 2 PLC library TcNcCfg.lib as string.

Outdated version

The sole purpose of this function is to ensure compatibility with existing projects. For new projects please use the global structure stLibVersion_Tc2_NCI.

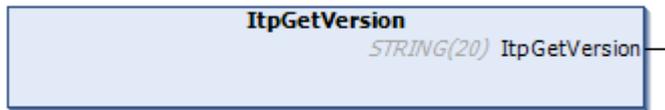
Return value

Get_TcNcCfg_Version Version number

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

6.1.5.3 ItpGetVersion



ItpGetVersion is a function that returns the version number of the TwinCAT PLC library TcNC.lib as string.

● Outdated version

i The sole purpose of this function is to ensure compatibility with existing projects. For new projects please use the global structure stLibVersion_Tc2_NCI.

VAR_INPUT

```
FUNCTION ItpGetVersion
VAR_INPUT
END_VAR
```

Return value

ItpGetVersion: Version number

Sample

```
VAR
    strVersion: STRING(20);
END_VAR
strVersion := ItpGetVersion();
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_NCI

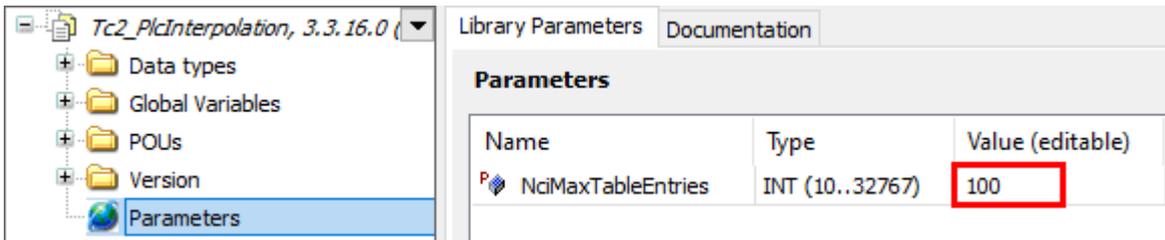
6.2 PLC Library: Tc2_PlcInterpolation

The Tc2_PlcInterpolation library offers an alternative to the application of G-Code (DIN 66025). This library can be used to execute interpolated movement commands directly from the PLC, without using G-Code.

In a first step a table of different movement commands and additional functions is written. To this end structures such as ST_NciGeoLine are transferred to the FB NciFeedTablePreparation. This appends the movement command to the table. Once the table is full or all required entries have been added, NciFeedTable is called in order to transfer the table content to the NC kernel. The data transfer directly starts the execution.

● NciMaxTableEntries can be edited

i From library version 3.3.16.0 (included from TC3.1.4024.11) the maximum number of table entries can be edited in the range from 10 to 32767. Default value is 100 entries.



Function blocks that are required for grouping of axes (or for channel control (channel override) can be found in the PLC Library: Tc2_NCI [▶ 195].

Function Block	Description
FB_NciFeedTablePreparation [▶ 296]	Fills a table with NCI movements in the PLC
FB_NciFeedTable [▶ 297]	Transfers a previously written table to the NC kernel and starts the motion

The following structures can be used as input parameters for the function block NciFeedTablePreparation:

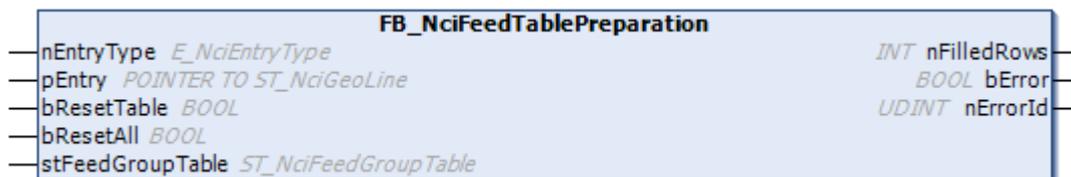
Structures	Enum	Description
Organization		
	E_NciEntryTypeNone	No function
ST_NciGeoStart [▶ 299]	E_NciEntryTypeGeoStart	Sets the start position for the first geometry entry
ST_NciEndOfTables [▶ 310]	E_NciEntryTypeEndOfTables	Indicates the end of the geometry table
Movement commands		
ST_NciGeoLine [▶ 300]	E_NciEntryTypeGeoLine	Describes a straight line
ST_NciGeoCirclePlane [▶ 300]	E_NciEntryTypeGeoCirclePlane	Describes a circle in the main plane (center point programming)
ST_NciGeoCircleCIP [▶ 302]	E_NciEntryTypeGeoCircleCIP	Describes a circle anywhere in the space
ST_NciGeoBezier3 [▶ 302]	E_NciEntryTypeGeoBezier3	Describes a 3rd order Bezier with control points
ST_NciGeoBezier5 [▶ 303]	E_NciEntryTypeGeoBezier5	Describes a 5th order Bezier with control points
ST_NciDwellTime [▶ 308]	E_NciEntryTypeDwellTime	Describes a dwell time
Path parameters		
ST_NciBaseFrame [▶ 307]	E_NciEntryTypeBaseFrame	Describes a zero shift and rotation
ST_NciVertexSmoothing [▶ 307]	E_NciEntryTypeVertexSmoothing	Activates blending at segment transitions
ST_NciTangentialFollowingDesc [▶ 309]	E_NciEntryTypeTfDesc	Activates tangential following of the tool
Dynamics		
ST_NciDynOvr [▶ 306]	E_NciEntryTypeDynOvr	Modifies the dynamic override
ST_NciAxisDynamics [▶ 308]	E_NciEntryTypeAxisDynamics	Limits the axis dynamics
ST_NciPathDynamics [▶ 308]	E_NciEntryTypePathDynamics	Limits the path dynamics
ST_NciFeedrateIpol [▶ 309]	E_NciEntryTypeFeedrateIpol	Sets the feed interpolation type
Parameter commands		
ST_NciHParam [▶ 305]	E_NciEntryTypeHParam	Sets an H-parameter (DINT)
ST_NciSParam [▶ 306]	E_NciEntryTypeSParam	Sets an S-parameter (WORD)
ST_NciTParam [▶ 306]	E_NciEntryTypeTParam	Sets a T-parameter (WORD)
ST_NciMFuncFast [▶ 305]	E_NciEntryTypeMFuncFast	Parameterizes a fast M-function (no handshake)

Structures	Enum	Description
ST_NciMFuncHsk [▶_304]	E_NciEntryTypeMFuncHsk	Parameterizes an M-function with handshake
ST_NciMFuncResetAllFast [▶_305]	E_NciEntryTypeResetAllFast	Resets all fast M-functions

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_PlcInterpolation

6.2.1 FB_NciFeedTablePreparation



The function block FB_NciFeedTablePreparation appends an entry of a specific type to the feed table (stFeedGroupTable). An appended entry can generate more than one row in the table. If the table has not enough free rows, an error is returned and no entry is added to the table. In this case the entry either has to be added to another table or to the same table, after FB_NciFeedTable was executed. This function block deals with modal functions, such as tangential following. It is therefore important to always use the same instance of this function block. The function block can be called repeatedly in a PLC cycle.

VAR_INPUT

```
VAR_INPUT
    nEntryType      : E_NciEntryType;
    pEntry          : POINTER TO ST_NciGeoLine;
    bResetTable     : BOOL;
    bResetAll       : BOOL;
END_VAR
```

nEntryType: Specifies the entry type, e.g. line, circle, tangential following

pEntry: Pointer to entry structure – must match nEntryType

bResetTable: If bResetTable is TRUE, the table 'stFeedGroupTable' is set to zero and nFilledRows is also set to zero. If nErrorId = ErrNciFeedTableFull, this error is reset. All modal flags (such as tangential following) remain constant.

bResetAll: Like bResetTable. In addition, all modal flags are set to their default values, and all error IDs are reset.

VAR_IN_OUT

```
VAR_IN_OUT
    stFeedGroupTable : ST_NciFeedGroupTable
END_VAR
```

stFeedGroupTable: Table containing the rows for the NC kernel.

VAR_OUTPUT

```
VAR_OUTPUT
    nFilledRows      : INT;
    bError           : BOOL;
    nErrorId         : UDINT;
END_VAR
```

nFilledRows: Number of filled rows.

bError: Becomes TRUE as soon as an error has occurred.

nErrorId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation (error codes above 0x4000).

Note If **bResetTable**, **bResetAll**, or **bError** is true, no further entries are accepted.

Note The error code 0x4B72 indicates that the table is full and the last entry was not accepted.

Example:

```
stGeoLine.nDisplayIndex := 1;
stGeoLine.fEndPosX := 0;
stGeoLine.fEndPosY := 400;
stGeoLine.fEndPosZ := 100;
stGeoLine.fEndPosQ1 := -90;
stGeoLine.fVelo := 1000; (*mm per sec*)

fbFeedTablePrep (
  nEntryType := E_NciEntryTypeGeoLine,
  pEntry := ADR(stGeoLine),
  bResetTable:= FALSE,
  stFeedGroupTable:= stNciFeedGroupTable,
  nFilledRows=> nFilledRows,
  bError => bError,
  nErrorId => nErrorId);
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_PlcInterpolation

6.2.2 FB_NciFeedTable



The function block **FB_NciFeedTable** transfers a given table to the NC kernel. If the override is set and the approvals are enabled, execution is started immediately. **bFeedingDone** becomes TRUE when the transfer is complete. This signal can be used for overwriting the table with NciFeedTablePreparation [▶ 296]. In NciFeedTablePreparation the table first has to be reset.

bChannelDone indicates complete execution of the tables in the NC kernel. The identifier ST_NciEndOfTables [▶ 298] must therefore be placed at the end of the last table.

VAR_INPUT

```
VAR_INPUT
  bExecute      : BOOL;
  bReset        : BOOL;
  bLogFeederEntries : BOOL;
END_VAR
```

bExecute: The command is triggered by a rising edge at this input.

bReset: Triggers a channel reset and also resets the function block

bLogFeederEntries: If TRUE, a log file 'PlcItpFeed.log' is written in the TwinCAT\Mc\Nci folder. It contains all entries that are sent to the NC kernel via ADS. If **bLogFeederEntries** = TRUE, more time is required until **bFeedingDone** becomes TRUE.

VAR_IN_OUT

```

VAR_IN_OUT
  stFeedGroupTable : ST_NciFeedGroupTable;
  stNciToPlc       : NCTOPLC_NCICHANNEL_REF;
END_VAR

```

stFeedGroupTable: Table containing the rows for the NC kernel.

stNciToPlc: The structure of the cyclic channel interface between NCI and PLC.

VAR_OUTPUT

```

VAR_OUTPUT
  bFeedingDone      : BOOL;
  bChannelDone      : BOOL;
  bFeedBusy         : BOOL;
  bResetBusy        : BOOL;
  bError            : BOOL;
  nErrorId          : UDINT;
END_VAR

```

bFeedingDone: Becomes TRUE once all table rows have been sent to the NC kernel.

bChannelDone: Becomes TRUE once all entries of the table in the NC kernel were executed and ST_NciEndOfTables was detected.

bFeedBusy: Becomes TRUE when the function block sends entries to the NC kernel.

bResetBusy: Becomes TRUE when a reset is executed.

bError: Becomes TRUE as soon as an error has occurred.

nErrorId: Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs. The error numbers in ErrId can be looked up in the ADS error documentation or in the NC error documentation.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1.0	PC or CX (x86 or x64)	Tc2_PlcInterpolation

6.2.3 Types and Enums

E_NciEntryType

```

TYPE E_NciEntryType :
(
  E_NciEntryTypeNone := 0,
  E_NciEntryTypeGeoStart := 1,
  E_NciEntryTypeGeoLine := 2,
  E_NciEntryTypeGeoCirclePlane := 3,
  E_NciEntryTypeGeoCircleCIP := 4,
  E_NciEntryTypeGeoBezier3 := 10,
  E_NciEntryTypeGeoBezier5 := 11,
  E_NciEntryTypeMFuncHsk := 20,
  E_NciEntryTypeMFuncFast := 21,
  E_NciEntryTypeMFuncResetAllFast := 23,
  E_NciEntryTypeHParam := 24,
  E_NciEntryTypeSParam := 25,
  E_NciEntryTypeTParam := 26,
  E_NciEntryTypeDynOvr := 50,
  E_NciEntryTypeVertexSmoothing := 51,
  E_NciEntryTypeBaseFrame := 52,
  E_NciEntryTypePathDynamics := 53,
  E_NciEntryTypeAxisDynamics := 55,
  E_NciEntryTypeDwellTime := 56,
  E_NciEntryTypeFeedrateIpol := 57,
  E_NciEntryTypeTfDesc := 100,
  E_NciEntryTypeEndOfTables := 1000
);
END_TYPE

```

Structures	Enum	Description
Organization		
	E_NciEntryTypeNone	No function
ST_NciGeoStart [▶ 299]	E_NciEntryTypeGeoStart	Sets the start position for the first geometry entry
ST_NciEndOfTables [▶ 310]	E_NciEntryTypeEndOfTables	Indicates the end of the geometry table
Movement commands		
ST_NciGeoLine [▶ 300]	E_NciEntryTypeGeoLine	Describes a straight line
ST_NciGeoCirclePlane [▶ 300]	E_NciEntryTypeGeoCirclePlane	Describes a circle in the main plane (center point programming)
ST_NciGeoCircleCIP [▶ 302]	E_NciEntryTypeGeoCircleCIP	Describes a circle anywhere in the space
ST_NciGeoBezier3 [▶ 302]	E_NciEntryTypeGeoBezier3	Describes a 3rd order Bezier with control points
ST_NciGeoBezier5 [▶ 303]	E_NciEntryTypeGeoBezier5	Describes a 5th order Bezier with control points
ST_NciDwellTime [▶ 308]	E_NciEntryTypeDwellTime	Describes a dwell time
Path parameters		
ST_NciBaseFrame [▶ 307]	E_NciEntryTypeBaseFrame	Describes a zero shift and rotation
ST_NciVertexSmoothing [▶ 307]	E_NciEntryTypeVertexSmoothing	Activates blending at segment transitions
ST_NciTangentialFollowingDesc [▶ 309]	E_NciEntryTypeTfDesc	Activates tangential following of the tool
Dynamics		
ST_NciDynOvr [▶ 306]	E_NciEntryTypeDynOvr	Modifies the dynamic override
ST_NciAxisDynamics [▶ 308]	E_NciEntryTypeAxisDynamics	Limits the axis dynamics
ST_NciPathDynamics [▶ 308]	E_NciEntryTypePathDynamics	Limits the path dynamics
ST_NciFeedrateIpol [▶ 309]	E_NciEntryTypeFeedrateIpol	Sets the feed interpolation type
Parameter commands		
ST_NciHParam [▶ 305]	E_NciEntryTypeHParam	Sets an H-parameter (DINT)
ST_NciSParam [▶ 306]	E_NciEntryTypeSParam	Sets an S-parameter (WORD)
ST_NciTParam [▶ 306]	E_NciEntryTypeTParam	Sets a T-parameter (WORD)
ST_NciMFuncFast [▶ 305]	E_NciEntryTypeMFuncFast	Parameterizes a fast M-function (no handshake)
ST_NciMFuncHsk [▶ 304]	E_NciEntryTypeMFuncHsk	Parameterizes an M-function with handshake
ST_NciMFuncResetAllFast [▶ 305]	E_NciEntryTypeResetAllFast	Resets all fast M-functions

ST_NciGeoStart

Sets the start position for the first geometry entry. This is necessary, if the first geometry entry is a circle or if tangential following in the first segment is ON. This structure can optionally be written at each start of the first table.

```

TYPE ST_NciGeoStart :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeGeoStart; (*do not override this parameter *)
  fPosX: LREAL;
  fPosY: LREAL;
  fPosZ: LREAL;
  fPosQ1: LREAL;
  fPosQ2: LREAL;
  fPosQ3: LREAL;
  fPosQ4: LREAL;

```

```
fPosQ5: LREAL;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

fPosX: Start position X

fPosY: Start position Y

fPosZ: Start position Z

fPosQ1: Start position Q1

fPosQ2: Start position Q2

fPosQ3: Start position Q3

fPosQ4: Start position Q4

fPosQ5: Start position Q5

ST_NciGeoLine

Describes a straight line with specified velocity.

```
TYPE ST_NciGeoLine :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeGeoLine; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  fEndPosX: LREAL;
  fEndPosY: LREAL;
  fEndPosZ: LREAL;
  fEndPosQ1: LREAL;
  fEndPosQ2: LREAL;
  fEndPosQ3: LREAL;
  fEndPosQ4: LREAL;
  fEndPosQ5: LREAL;
  fVelo: LREAL;
  bRapidTraverse: BOOL;
  bAccurateStop: BOOL; (* VeloEnd := 0 *)
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fEndPosX: Target position X

fEndPosY: Target position Y

fEndPosZ: Target position Z

fEndPosQ1: Target position Q1

fEndPosQ2: Target position Q2

fEndPosQ3: Target position Q3

fEndPosQ4: Target position Q4

fEndPosQ5: Target position Q5

fVelo: Target path velocity, like F in G-Code, but in basic units per second (e.g. mm/s)

bRapidTraverse: TRUE has the same effect as G0, FALSE treats this entry like G01

bAccurateStop: Accurate stop (TRUE has the same effect as G09)

ST_NciGeoCirclePlane

Describes a circle in the main plane. The center point is specified in absolute coordinates.

The orthogonal component at the center is assigned internally. If a circle is programmed in the XY plane, for example, `fCenterZ` is assigned internally. If the user has assigned the value explicitly, the value is nevertheless overwritten by the function block. A helix can be described by programming the height. If helix is programmed in the XY plane, for example, the lifting height of the helix is specified absolutely with `fEndPosZ`.

```

TYPE ST_NciGeoCirclePlane :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeGeoCirclePlane; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  fEndPosX: LREAL;
  fEndPosY: LREAL;
  fEndPosZ: LREAL;
  fCenterX: LREAL;
  fCenterY: LREAL;
  fCenterZ: LREAL;
  fEndPosQ1: LREAL;
  fEndPosQ2: LREAL;
  fEndPosQ3: LREAL;
  fEndPosQ4: LREAL;
  fEndPosQ5: LREAL;
  fVelo: LREAL;
  bClockwise: BOOL;
  bAccurateStop: BOOL; (* VeloEnd := 0 *)
  nPlane: E_NciGeoPlane := E_NciGeoPlaneXY;
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fEndPosX: Target position X

fEndPosY: Target position Y

fEndPosZ: Target position Z

fCenterX: Centre position X in absolute coordinates

fCenterY: Centre position Y in absolute coordinates

fCenterZ: Centre position Z in absolute coordinates

fEndPosQ1: Target position Q1

fEndPosQ2: Target position Q2

fEndPosQ3: Target position Q3

fEndPosQ4: Target position Q4

fEndPosQ5: Target position Q5

fVelo: Target path velocity in basic units per second (e.g. mm/s), like F in G-Code

bClockwise: If TRUE, the circle is drawn clockwise, otherwise counter-clockwise (similar to G02, G03)

bAccurateStop: [accurate stop \[► 138\]](#) (TRUE has the same effect as G09)

nPlane: Specifies the plane: XY, YZ, or ZX (similar to G17..G19) (type: [E_NciGeoPlane \[► 301\]](#))

● Circle segment as start segment

I If the first geometry segment is a circle, the start position must set with [ST_NciGeoStart \[► 299\]](#).

E_NciGeoPlane

```

TYPE E_NciGeoPlane :
(
  E_NciGeoPlaneXY := 17,
  E_NciGeoPlaneZX := 18,

```

```

    E_NciGeoPlaneYZ := 19
);
END_TYPE

```

ST_NciGeoCircleCIP

The CIP circle can be used to describe a circle anywhere in space. It does not have to be in the main plane. In order for the circle to be described unambiguously, not all 3 points (the starting point is specified implicitly) may lie on straight line. It is thus not possible to program a full circle in this way.

```

TYPE ST_NciGeoCircleCIP :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeGeoCircleCIP;    (* do not overwrite this parameter
*)
  nDisplayIndex: UDINT;
  fEndPosX:      LREAL;
  fEndPosY:      LREAL;
  fEndPosZ:      LREAL;
  fCIPPosX:      LREAL;
  fCIPPosY:      LREAL;
  fCIPPosZ:      LREAL;
  fEndPosQ1:     LREAL;
  fEndPosQ2:     LREAL;
  fEndPosQ3:     LREAL;
  fEndPosQ4:     LREAL;
  fEndPosQ5:     LREAL;
  fVelo:         LREAL;
  bAccurateStop: BOOL;    (* VeloEnd := 0 *)
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType](#) [[▶ 298](#)])

nDisplayIndex: For display purposes, such as block number in G-Code

fCIPPosX: X position in absolute coordinates (point on circular path)

fCIPPosY: Y position in absolute coordinates (point on circular path)

fCIPPosZ: Z position in absolute coordinates (point on circular path)

fEndPosX: Target position X

fEndPosY: Target position Y

fEndPosZ: Target position Z

fEndPosQ1: Target position Q1

fEndPosQ2: Target position Q2

fEndPosQ3: Target position Q3

fEndPosQ4: Target position Q4

fEndPosQ5: Target position Q5

fVelo: Target path velocity in basic units per second (e.g. mm/s), like F in G-Code

bAccurateStop: [accurate stop](#) [[▶ 138](#)] (TRUE has the same effect as G09)

● Circle segment as start segment

I If the first geometry segment is a circle, the start position must set with [ST_NciGeoStart](#) [[▶ 299](#)].

ST_NciGeoBezier3

Describes a third-order Bézier curve with the aid of control points. The start position results from the previous segment. The third control point is determined by the target position.

```

TYPE ST_NciGeoBezier3:
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeGeoBezier3; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  fControlPoint1X: LREAL;
  fControlPoint1Y: LREAL;
  fControlPoint1Z: LREAL;
  fControlPoint2X: LREAL;
  fControlPoint2Y: LREAL;
  fControlPoint2Z: LREAL;
  fEndPosX: LREAL;
  fEndPosY: LREAL;
  fEndPosZ: LREAL;
  fEndPosQ1: LREAL;
  fEndPosQ2: LREAL;
  fEndPosQ3: LREAL;
  fEndPosQ4: LREAL;
  fEndPosQ5: LREAL;
  fVelo: LREAL;
  bAccurateStop: BOOL; (* VeloEnd := 0 *)
END_STRUCT
END_TYPE

```



A Bezier3 curve is not compatible with the type ST_NciVertexSmoothing of the type 3rd and 5th order Bezier. In this case, a different type must be selected for VertexSmoothing.

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fControlPoint1X: X component control point 1

fControlPoint1Y: Y component control point 1

...

fControlPoint2Z: Z component control point 2

fEndPosX: Target position X

fEndPosY: Target position Y

fEndPosZ: Target position Z

fEndPosQ1: Target position Q1

fEndPosQ2: Target position Q2

fEndPosQ3: Target position Q3

fEndPosQ4: Target position Q4

fEndPosQ5: Target position Q5

fVelo: Target path velocity in basic units per second (e.g. mm/s), like F in G-Code

bAccurateStop: [Accurate stop \[► 138\]](#) (TRUE has the same effect as G09)

ST_NciGeoBezier5

Describes a 5th-order Bézier curve with the aid of control points. The start position results from the previous segment. The fifth control point is determined by the target position.

```

TYPE ST_NciGeoBezier5:
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeGeoBezier5; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  fControlPoint1X: LREAL;
  fControlPoint1Y: LREAL;
  fControlPoint1Z: LREAL;
  fControlPoint2X: LREAL;

```

```

fControlPoint2Y: LREAL;
fControlPoint2Z: LREAL;
fControlPoint3X: LREAL;
fControlPoint3Y: LREAL;
fControlPoint3Z: LREAL;
fControlPoint4X: LREAL;
fControlPoint4Y: LREAL;
fControlPoint4Z: LREAL;
fEndPosX: LREAL;
fEndPosY: LREAL;
fEndPosZ: LREAL;
fEndPosQ1: LREAL;
fEndPosQ2: LREAL;
fEndPosQ3: LREAL;
fEndPosQ4: LREAL;
fEndPosQ5: LREAL;
fVelo: LREAL;
bAccurateStop: BOOL; (* VeloEnd := 0 *)
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fControlPoint1X: X component control point 1

fControlPoint1Y: Y component control point 1

...

fControlPoint4Z: Z component control point 4

fEndPosX: Target position X

fEndPosY: Target position Y

fEndPosZ: Target position Z

fEndPosQ1: Target position Q1

fEndPosQ2: Target position Q2

fEndPosQ3: Target position Q3

fEndPosQ4: Target position Q4

fEndPosQ5: Target position Q5

fVelo: Target path velocity in basic units per second (e.g. mm/s), like F in G-Code

bAccurateStop: [accurate stop \[► 138\]](#) (TRUE has the same effect as G09)

ST_NciMFuncHsk

Describes an [M-function \[► 161\]](#) of type handshake. The M-function number is between 0 and 159.

```

TYPE ST_NciMFuncHsk :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeMFuncHsk; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  nMFunc: INT;
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nMFunc: M-function number (0..159)

i M-functions in the PlcInterpolation library

If M-functions are used in the PlcInterpolation library, they do not have to be entered in the user interface of the XAE. An M-function always takes effect at the programmed location.

ST_NciMFuncFast

Parameterizes up to 8 fast M-functions [► 161]. The first M-function must be assigned nMFuncIn0, the second nMFuncIn1 etc. -1 indicates the end of the assignments.

```
TYPE ST_NciMFuncFast :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeMFuncFast; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  nMFuncIn0: INT;
  nMFuncIn1: INT;
  nMFuncIn2: INT;
  nMFuncIn3: INT;
  nMFuncIn4: INT;
  nMFuncIn5: INT;
  nMFuncIn6: INT;
  nMFuncIn7: INT;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nMFuncIn0: fast M-function number (0..159)

nMFuncIn1: Fast M-function number (0..159); -1 indicates the end of the list.

nMFuncIn2: Fast M-function number (0..159); -1 indicates the end of the list.

nMFuncIn3: Fast M-function number (0..159); -1 indicates the end of the list.

nMFuncIn4: Fast M-function number (0..159); -1 indicates the end of the list.

nMFuncIn5: Fast M-function number (0..159); -1 indicates the end of the list.

nMFuncIn6: Fast M-function number (0..159); -1 indicates the end of the list.

nMFuncIn7: Fast M-function number (0..159); -1 indicates the end of the list.

i M-functions in the PlcInterpolation library

If M-functions are used in the PlcInterpolation library, they do not have to be entered in the user interface of the XAE. An M-function always takes effect at the programmed location.

ST_NciMFuncResetAllFast

Resets all fast M-functions [► 161].

```
TYPE ST_NciMFuncResetAllFast :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeMFuncResetAllFast; (*do not override this parameter *)
  nDisplayIndex: UDINT;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

ST_NciHParam

Sets an [H-parameter \[► 165\]](#) in the cyclic channel interface.

```

TYPE ST_NciHParam :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeHParam; (*do not override this parameter *)
    nDisplayIndex: UDINT;
    nHParam: UDINT;
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nHParam: H-parameter from NC to PLC

ST_NciSParam

Sets an [S-parameter \[► 165\]](#) in the cyclic channel interface.

```

TYPE ST_NciSParam :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeSParam; (*do not override this parameter *)
    nDisplayIndex: UDINT;
    nSParam: UINT;
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nSParam: S-parameter from NC to PLC

ST_NciTParam

Sets an [T-parameter \[► 165\]](#) in the cyclic channel interface.

```

TYPE ST_NciTParam :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeTParam; (*do not override this parameter *)
    nDisplayIndex: UDINT;
    nTParam: UINT;
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nTParam: T-parameter from NC to PLC

ST_NciDynOvr

Modal functions for changing the path dynamics.

See [DynOvr \[► 172\]](#) in the [interpreter documentation \[► 122\]](#).

```

TYPE ST_NciDynOvr :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeDynOvr; (*do not override this parameter*)
    nDisplayIndex: UDINT;
    fDynOvr: LREAL;
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fDynOvr: Value for dynamic override ($1 < fDynOvr \leq 1$)

ST_NciVertexSmoothing

Modal function for activating blending at the segment transition. Blending is active until it is cancelled by setting the radius to 0.

A more detailed description of the parameter can be found in the [interpreter documentation \[► 122\]](#). ([paramVertexSmoothing \[► 149\]](#)).

```
TYPE ST_NciVertexSmoothing :
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeVertexSmoothing; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  nType: UDINT; (*type of smoothing, e.g. parabola, bi-quad *)
  nSubtype: UDINT; (*e.g. adaptive, constant radius *)
  fRadius: LREAL; (*max. radius for tolerance ball *)
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nType: Blending type: 2: parabola, 3: Bi-quadratic, 4: Bezier 3rd order, 5: 5th order Bezier

nSubtype: 1: constant tolerance radius, 2: distance between intersection and vertex, 3: Adaptive tolerance radius

fRadius: Radius of the blending sphere in basic units (e.g. mm)

ST_NciBaseFrame

The structure ST_NciBaseFrame describes a modal zero shift and rotation. The operating principle is the same as for zero shift and rotation in the interpreter, i.e. the point of rotation is the current origin (see [rotation \[► 145\]](#) in the [interpreter documentation \[► 122\]](#)).

```
TYPE ST_NciBaseFrame:
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeBaseFrame; (*Do not override this parameter *)
  nDisplayIndex: UDINT;
  fShiftX: LREAL;
  fShiftY: LREAL;
  fShiftZ: LREAL;
  fRotX: LREAL;
  fRotY: LREAL;
  fRotZ: LREAL;
  fShiftQ1: LREAL;
  fShiftQ2: LREAL;
  fShiftQ3: LREAL;
  fShiftQ4: LREAL;
  fShiftQ5: LREAL;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[► 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fShiftX: Zero shift in X direction

fShiftY: Zero shift in Y direction

fShiftZ: Zero shift in Z direction

fRotX: Rotation of the X axis

fRotY: Rotation of the Y axis

fRotZ: Rotation of the Z axis

fShiftQ1: Offset of the Q1 axis

fShfitQ2: Offset of the Q2-axis

fShiftQ3: Offset of the Q3-axis

fShiftQ4: Offset of the Q4-axis

fShiftQ5: Offset of the Q5-axis

ST_NciPathDynamics

The structure *ST_NciPathDynamics* sets the path dynamics (acceleration, deceleration, jerk). The operating principle is the same as for *paramPathDynamics* in the interpreter (see [paramPathDynamics \[▸ 172\]](#) in the [interpreter documentation \[▸ 122\]](#)).

```
TYPE ST_NciPathDynamics:
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypePathDynamics; (*do not override this parameter *)
  nDisplayIndex: UDINT;
  fAcc: LREAL;
  fDec: LREAL;
  fJerk: LREAL;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[▸ 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fAcc: Maximum permitted path acceleration

fDec: Maximum permitted path deceleration

fJerk: Maximum permitted path jerk

ST_NciAxisDynamics

The structure *ST_NciAxisDynamics* sets the path axis dynamics (acceleration, deceleration, jerk). The operating principle is the same as for *paramAxisDynamics* in the interpreter (see [paramAxisDynamics \[▸ 172\]](#) in the [interpreter documentation \[▸ 122\]](#)).

```
TYPE ST_NciAxisDynamics:
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeAxisDynamics; (*Do not override this parameter*)
  nDisplayIndex: UDINT;
  nAxis: UDINT;
  fAcc: LREAL;
  fDec: LREAL;
  fJerk: LREAL;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[▸ 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

nAxis: Axis in interpolation group X:0 Y:1 Z:2 Q1:3 ... Q5:7

fAcc: Maximum permitted axis acceleration

fDec: Maximum permitted axis deceleration

fJerk: Maximum permitted axis jerk

ST_NciDwellTime

The structure *ST_NciDwellTime* is used to activate a dwell time in seconds (see [dwell time \[▸ 137\]](#) in the [interpreter documentation \[▸ 122\]](#)).

```
TYPE ST_NciDwellTime:
STRUCT
  nEntryType: E_NciEntryType := E_NciEntryTypeDwellTime; (*Do not override this parameter *)
  nDisplayIndex: UDINT;
```

```
fDwellTime: LREAL;
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[▸ 298\]](#))

nDisplayIndex: For display purposes, such as block number in G-Code

fDwellTime: Dwell time in seconds

ST_NciFeedratelpol

The structure *ST_NciFeedratelpol* can be used to set the feed interpolation (see [Feed interpolation \[▸ 138\]](#)).

```
TYPE ST_NciFeedrateIpol :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeFeedrateIpol; (*Do not overwrite this parameter*)
    nDisplayIndex: UDINT;
    eFeedrateIpol: E_NciFeedrateIpol; (*E_NciFeedrateIpolConstant = FCONST,
E_NciFeedrateIpolLinear=FLIN *)
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[▸ 298\]](#))

eFeedratelpol: specifies the feed interpolation.

```
TYPE E_NciFeedRateIpol : (
    E_NciFeedrateIpolConstant,
    E_NciFeedrateIpolLinear
)
END_TYPE
```

ST_NciTangentialFollowingDesc

This is a modal command for switching tangential following on or off.

```
TYPE ST_NciTangentialFollowingDesc :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeTfDesc; (*do not override this parameter *)
    bTangOn: BOOL;
    nTangAxis: E_NciAxesInGroup; (*axis used for tangential following *)
    nPathAxis1: E_NciAxesInGroup; (*describing the plane e.g. x*)
    nPathAxis2: E_NciAxesInGroup; (*e.g. y ==> g17, xy plane*)
    fOffset: LREAL; (*geo tangent is 0 degree, counting is mathematical positive *)
    fCriticalAngle1: LREAL;
    nTfBehavior: E_TangentialFollowingBehavior; (*what to do if angle becomes bigger than critical
angle 1 *)
END_STRUCT
END_TYPE
```

nEntryType: Do not override this parameter (type: [E_NciEntryType \[▸ 298\]](#))

bTangOn: If TRUE, tangential following is switched on.

nTangAxis: Axis (Q1..Q5) that is used as tangential axis (type: [E_NciAxesInGroup \[▸ 309\]](#)).

nPathAxis1: First path axis describing the plane and orientation for calculating the tangent.

nPathAxis2: Second path axis describing the plane and orientation for calculating the tangent.

fOffset: Offset of the tangential axis

fCriticalAngle1: Critical angle 1. The response in cases where the angle between two segments is greater than fCriticalAngle1 is specified with nTfBehavior.

nTfBehavior: see fCriticalAngle1 (type: [E_TangentialFollowingBehavior \[▸ 310\]](#))

E_NciAxesInGroup

```
TYPE E_NciAxesInGroup :
(
    NoneAxis := 0,
    XAxis,
```

```

    YAxis,
    ZAxis,
    Q1Axis,
    Q2Axis,
    Q3Axis,
    Q4Axis,
    Q5Axis
);
END_TYPE

```

E_TangentialFollowingBehavior

```

TYPE E_TangentialFollowingBehavior :
(
    E_TfIgnoreAll, (*ignore critical angle *)
    E_TfErrorOnCritical1 (*if angle becomes bigger than critical angle 1 ==> error *)
);
END_TYPE

```

E_TfIgnoreAll: The critical angle is ignored.

E_TfErrorOnCritical1: An error is returned if the critical angle is exceeded.

ST_NciEndOfTables

Indicates the last entry of the last table. Is used for signaling the bChannelDone flag in [FB_NciFeedTable](#) [[▶ 297](#)].

```

TYPE ST_NciEndOfTables :
STRUCT
    nEntryType: E_NciEntryType := E_NciEntryTypeEndOfTables; (*do not override this parameter *)
END_STRUCT
END_TYPE

```

nEntryType: Do not override this parameter (type: [E_NciEntryType](#) [[▶ 298](#)])

7 Samples

NCI: NCISimpleSample

Download:

https://infosys.beckhoff.com/content/1033/TF5100_TC3_NC_I/Resources/3438746891/.zip

The example NCISimpleSample shows how an G-Code program is loaded from the PLC and processing is started.

You need to copy the enclosed parts program Testlt.nc into the TwinCAT\Mc\Nci directory. Otherwise the parts program will not be found during loading. Alternatively you can adjust the path in the PLC program.

PLC interpolation: PlcInterpolationSimpleSample

Download:

https://infosys.beckhoff.com/content/1033/TF5100_TC3_NC_I/Resources/2944140171/.zip

The sample shows how a movement can be affected with the library Tc2_PlcInterpolation directly from the PLC.

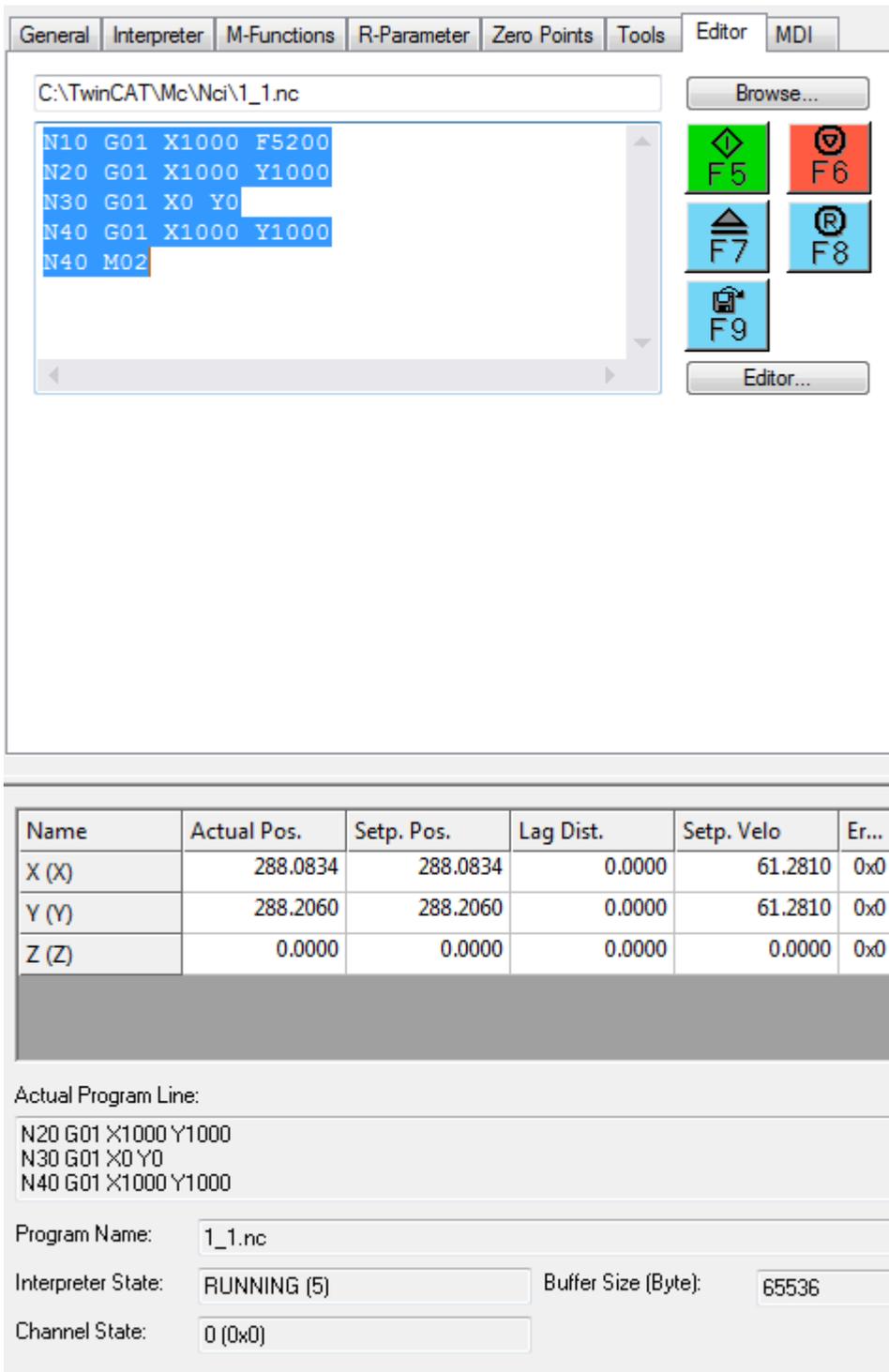
8 Appendix

8.1 Display of the parts program

Reading of the current NC line via ADS

This ADS Read command returns a maximum of three lines of the current parts program, i.e. the current line of code and perhaps two previously processed lines.

Function	ADS-Read
Port	500 (dec)
Index Group	0x2300 + channel ID
Index Offset	0x2000 0001
Data	string (30 bytes min.)



Reading of the current program name

This ADS Read command returns the program name of the current main NC program (in this case 1_1.nc).

Function	ADS-Read
Port	500 (dec)
Index Group	0x2100 + channel ID
Index Offset	0x7
Data	string, 100 characters max.

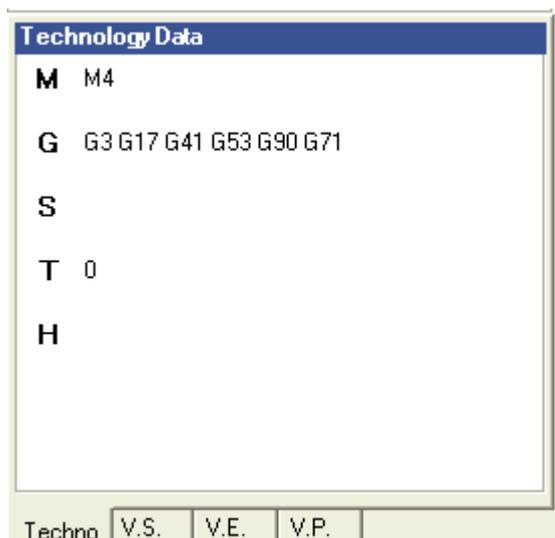
Reading of the current file information

In contrast to the 'Reading the current NC line' function, in this case not the line itself is read, but associated line information. The return value is the current program name (e.g. file name of the subroutine) and a file offset. Based on this information, the user interface can open the associated file and highlight the respective line. The display is no longer limited to 3 rows, i.e. any number of lines can be displayed.

In the event of an NCI load or runtime error, information about the associated line of code can be obtained via this route.

Function	ADS-Read	
Port	500 (dec)	
Index Group	0x2100 + channel ID	
Index Offset	0x12	
Data	UINT32	Current display of 1: SAF- 2: Interpreter 3: Error offset
	UINT32	File offset
	char[260]	path + program name

8.2 Display of technology data



The currently active technology data such as G functions, zero shifts and rotation can be read via ADS.

Activation for reading the technology data

In order to read the above-mentioned parameters, activation via ADS is required first.

The function must be activated before the start of the NC program, or earlier. It remains active until either a TwinCAT restart is performed or the function is reset explicitly.

Function	ADS-Write
Port	500 (dec)
Index Group	0x2000 + channel ID
Index Offset	0x0053
Data	DWORD 0: disable (default) 1: enable

Reading the currently active zero shift

This command reads the active zero shift of the segment currently in block execution (SAF). If no zero shift is active (G53), the structure for the individual components contains a zero vector. These data can be used for switching the display between machine coordinates and programming coordinates, for example.

The data, which are read with the function block 'ItpReadZeroShift', for example, may differ from these values, since the interpreter data are read with the function block, which may already take into account new offsets.

Function	ADS-Read	
Port	500 (dec)	
Index Group	0x2100 + channel ID	
Index Offset	0x0014	
Data	{	
	UINT32	block counter
	UINT32	dummy
	LREAL[3]	zero shift G54..G57
	LREAL[3]	zero shift G58
	LREAL[3]	zero shift G59
	}	

Reading the currently active rotation

This command reads the active rotation of the segment currently in block execution (SAF).

Function	ADS-Read	
Port	500 (dec)	
Index Group	0x2100 + channel ID	
Index Offset	0x0015	
Data	{	
	UINT32	block counter
	UINT32	dummy
	LREAL[3]	rotation of X, Y & Z in degrees
	}	

Reading the currently active G-Code

The G-Code is subdivided into groups. For example, the geometries types with modal effect (G01, G02...) and the plane selection (G17..G19) form separate groups. When the G-Code information is read, the enumerator for the groups is also read. These can then be displayed in an application-specific manner.

Since the read command comes with a parameter to be read, not all groups have to be read. The memory provided is always filled by group 1. If, for example, the transferred memory size is 3x8 bytes, the data for the block counter, group 1 and 2 are returned.

Function	ADS-Read	
Port	500 (dec)	
Index Group	0x2100 + channel ID	
Index Offset	0x0013	
Data	{	
	UINT32	block counter
	UINT32	Group 1: ModalGeoTypes
	UINT32	Group 2: BlockwiseGeoTypes
	UINT32	Group 3: ModalPlaneSelection
	UINT32	Group 4: ModalToolCompensation

UINT32	Group 5: ModalToolFeedDirection
UINT32	Group 6: ModalZeroShift
UINT32	Group 7: ModalAccurateStop
UINT32	Group 8: BlockwiseAccurateStop
UINT32	Group 9: ModalDesignationAbsInc
UINT32	Group 10: ModalDesignationInchMetric
UINT32	Group 11: ModalFeedRateInCurve
UINT32	Group 12: ModalCenterpointCorr
UINT32	Group 13: ModalCircleCpAbsInc
UINT32	Group 14: ModalCollisionDetection
UINT32	Group 15: ModalRotation
UINT32	Group 16: ModalCalcExRot
UINT32	Group 17: ModalDiam
UINT32	Group 18: ModalFeedrateIpol
UINT32	Group 19: ModalMirror
}	

```
#define GCodeOffset 0x1000
#define CommonIdentOffset 0x2000 // used for non-g-code commands, like rot, cfc...
```

Group 1: ModalGeoTypes

```
enum GCodeGroup_ModalGeoTypes
{
  ModalGeoTypeUndefined = 0,
  ModalGeoTypeG0 = 0 + GCodeOffset, // line - rapid traverse
  ModalGeoTypeG01 = 1 + GCodeOffset, // straight line
  ModalGeoTypeG02 = 2 + GCodeOffset, // circle clockwise
  ModalGeoTypeG03 = 3 + GCodeOffset // circle anticlockwise
};
```

Group 2: BlockwiseGeoTypes

```
enum GCodeGroup_BlockwiseGeoTypes
{
  BlockwiseGeoTypeNone = 0,
  BlockwiseGeoTypeG04 = 4 + GCodeOffset, // dwell time
  BlockwiseGeoTypeG74 = 74 + GCodeOffset, // homing
  BlockwiseGeoTypeCip = 1 + CommonIdentOffset // circle parametrized with 3 points
};
```

Group 3: ModalPlaneSelection

```
enum GCodeGroup_ModalPlaneSelection
{
  ModalPlaneSelectUndefined = 0,
  ModalPlaneSelectG17 = 17 + GCodeOffset, // xy-plane
  ModalPlaneSelectG18 = 18 + GCodeOffset, // zx-plane
  ModalPlaneSelectG19 = 19 + GCodeOffset // yz-plane
};
```

Group 4: ModalToolCompensation

```
enum GCodeGroup_ModalToolCompensation
{
  ModalToolCompUndefined = 0,
  ModalToolCompG40 = 40 + GCodeOffset, // tool compensation off
  ModalToolCompG41 = 41 + GCodeOffset, // tool compensation left
  ModalToolCompG42 = 42 + GCodeOffset // tool compensation right
};
```

Group 5: ModalToolFeedDirection

```
enum GCodeGroup_ModalToolFeedDirection
{
  ModalToolFeedDirUndefined = 0,
  ModalToolFeedDirPos = 2 + CommonIdentOffset, // tool feed direction positive
  ModalToolFeedDirNeg = 3 + CommonIdentOffset // tool feed direction negative
};
```

Group 6: ModalZeroShift

```
enum GCodeGroup_ModalZeroShift
{
ModalZeroShiftUndefined = 0,
ModalZeroShiftG53 = 53 + GCodeOffset, // zero shift off
ModalZeroShiftG54G58G59 = 54 + GCodeOffset, // zero shift G54 + G58+ G59
ModalZeroShiftG55G58G59 = 55 + GCodeOffset, // zero shift G55 + G58+ G59
ModalZeroShiftG56G58G59 = 56 + GCodeOffset, // zero shift G56 + G58+ G59
ModalZeroShiftG57G58G59 = 57 + GCodeOffset // zero shift G57 + G58+ G59
};
```

Group 7: ModalAccurateStop

```
enum GCodeGroup_ModalAccurateStop
{
ModalAccurateStopNone = 0,
ModalAccurateStopG60 = 60 + GCodeOffset // modal accurate stop
};
```

Group 8: BlockwiseAccurateStop

```
enum GCodeGroup_BlockwiseAccurateStop
{
BlockwiseAccurateStopNone = 0,
BlockwiseAccurateStopG09 = 9 + GCodeOffset, // common accurate stop
BlockwiseAccurateStopTpm = 4 + CommonIdentOffset // target position monitoring
};
```

Group 9: ModalDesignationAbsInc

```
enum GCodeGroup_ModalDesignationAbsInc
{
ModalDesignAbsIncUndefined = 0,
ModalDesignAbsIncG90 = 90 + GCodeOffset, // absolute designation
ModalDesignAbsIncG91 = 91 + GCodeOffset // incremental designation
};
```

Group 10: ModalDesignationInchMetric

```
enum
GCodeGroup_ModalDesignationInchMetric
{
ModalDesignInchMetricUndefined = 0,
ModalDesignInchMetricG70 = 70 + GCodeOffset, // designation inch
ModalDesignInchMetricG71 = 71 + GCodeOffset, // designation metric
ModalDesignInchMetricG700 = 700 + GCodeOffset, // designation inch & feedrate recalculated
ModalDesignInchMetricG710 = 710 + GCodeOffset // designation metric & feedrate recalculated
};
```

Group 11: ModalFeedRateInCurve

```
enum GCodeGroup_ModalFeedRateInCurve
{
ModalFeedRateInCurveUndefined = 0,
ModalFeedRateInCurveCfc = 5 + CommonIdentOffset, // constant feed contour
ModalFeedRateInCurveCfin = 6 + CommonIdentOffset, // constant feed inner contour
ModalFeedRateInCurveCftcp = 7 + CommonIdentOffset // constant feed tool center point
};
```

Group 12: ModalCenterpointCorr

```
enum GCodeGroup_ModalCenterpointCorr
{
ModalCenterpointCorrUndefined = 0,
ModalCenterpointCorrOn = 8 + CommonIdentOffset, // circle centerpoint correction on
ModalCenterpointCorrOff = 9 + CommonIdentOffset // circle centerpoint correction off
};
```

Group 13: ModalCircleCpAbsInc

```
enum GCodeGroup_ModalCircleCpAbsInc
{
ModalCircleCpUndefined = 0,
ModalCircleCpIncremental = 10 + CommonIdentOffset, // circle centerpoint incremental to start point
ModalCircleCpAbsolute = 11 + CommonIdentOffset // circle centerpoint absolute
};
```

Group 14: ModalCollisionDetection

```
enum GCodeGroup_ModalCollisionDetection
{
ModalCollisionDetectionUndefined = 0,
ModalCollisionDetectionOn = 12 + CommonIdentOffset, //collision detection on
ModalCollisionDetectionOff = 13 + CommonIdentOffset //collision detection off
};
```

Group 15: ModalRotation

```
enum GCodeGroup_ModalRotation
{
ModalRotationUndefined = 0,
ModalRotationOn = 14 + CommonIdentOffset, // rotation is turned on
ModalRotationOff = 15 + CommonIdentOffset // rotation is turned off
};
```

Group 16: ModalCalcExRot

```
enum GCodeGroup_ModalCalcExRot
{
ModalCalcExRotUndefined = 0,
ModalCalcExRotOn = 16 + CommonIdentOffset, // extended calculation for rotation turned on
ModalCalcExRotOff = 17 + CommonIdentOffset // extended calculation for rotation turned off
};
```

Group 17: ModalDiam

```
enum GCodeGroup_ModalDiam
{
ModalDiamUndefined = 0,
ModalDiamOn = 18 + CommonIdentOffset, // diameter programming on
ModalDiamOff = 19 + CommonIdentOffset // diameter programming off
};
```

Group 18: ModalFeedrateIpol

```
enum GCodeGroup_ModalFeedrateIpol
{
ModalFeedrateIpolUndefined = 0,
ModalFeedrateIpolConst = 20 + CommonIdentOffset, // federate interpolation constant (default)
ModalFeedrateIpolLinear = 21 + CommonIdentOffset // federate interpoaltion linear to remaining path
};
```

Group 19: ModalMirror

```
enum GCodeGroup_ModalMirror
{
// value - (32+CommonIdentOffset) shows the bitmask for mirrored axes
// that's why the sequence seems to be strange...
//
ModalMirrorUndefined = 0,
ModalMirrorOff = 32 + CommonIdentOffset,
ModalMirrorX = 33 + CommonIdentOffset,
ModalMirrorY = 34 + CommonIdentOffset,
ModalMirrorXY = 35 + CommonIdentOffset,
ModalMirrorZ = 36 + CommonIdentOffset,
ModalMirrorZX = 37 + CommonIdentOffset,
ModalMirrorYZ = 38 + CommonIdentOffset,
ModalMirrorXYZ = 39 + CommonIdentOffset
};
```

8.3 Displaying the remaining path length

If calculation of the remaining path length is switched active, it is calculated up to as far as the next accurate stop, or as far as the last geometric segment in memory (block preparation). An accurate stop is, for instance, generated by G09 or by G60. However, M-functions of type handshake, decoder stops and G04 implicitly generate an accurate stop.

Activation:

Index Group: 0x3000 + Group ID
Index Offset: 0x0508

see index offset specification for group parameters

Reading the remaining path length:

Reading is again implemented through ADS, and can also be recorded with TwinCAT Scope.

Index Group: 0x3100 + Group ID
 Index Offset: 0x0522

The remaining path length can be transferred with the cyclic channel interface to the PLC via [ltpSetCyclicLrealOffsets](#) [[▶ 235](#)].
 see index offset specification for group state

8.4 Parameterisation

The parameterization of the NCI comprises the standard dynamic parameters (acceleration, deceleration, jerk) and their online changes, along with the minimum velocity and the parameters for the reduction of the path velocity including online change.

General characteristics at segment transitions

- Velocity: The segment set velocity VS changes at the segment transition from VS_in to VS_out. At the segment transition the velocity is always reduced to the lower of the two values.
- Acceleration: The current path acceleration is always returned to $a = 0$ at segment transition.
- Jerk: The jerk unit J changes according to the geometry at the segment transition. This can cause a significant step change in dynamics.
- It is possible to [smooth segment transitions](#) [[▶ 125](#)].

Table 1: NCI group parameters

Parameter	Meaning and boundary conditions
Curve velocity reduction mode [▶ 320]	Coulomb, cosine or VELOJUMP
Minimum velocity [▶ 319]	Path velocity which may not be less than this value (except peaks with movement reversal): $V_{min} \geq 0.0$
Reduction method for C1 transitions [▶ 320]	Reduction factor for C1 transitions: $C1 \geq 0.0$
VELOJUMP: C0 reduction factors C0X, C0Y, C0Z	Reduction factors for C0 transitions for X, Y, Z axis: $C0X \geq 0.0, C0Y \geq 0.0, C0Z \geq 0.0$ (axis parameters, online modification in interpreter [▶ 173] possible).
DEVIATIONANGLE: Reduction factor C0 C0	Path reduction factor for C0 transitions: $1.0 \geq C0 \geq 0.0$
DEVIATIONANGLE: Critical angle (low) φ_l	Angle from which a velocity reduction is applied at the segment transition: $0 \leq \varphi_l < \varphi_h \leq \pi$
DEVIATIONANGLE: Critical angle (high) φ_h	Angle from which the velocity at the segment transition (v_{link}) is reduced to 0.0: $0 \leq \varphi_l < \varphi_h \leq \pi$
Tolerance sphere radius [▶ 152] TBR	Radius of the tolerance spheres: $1000.0 \text{ mm} \geq TBR \geq 0.1 \text{ mm}$
C2 reduction factor [▶ 173] C2	Reduction factor for smoothed transitions: $C2 \geq 0.0$
Global software limit positions for the path [▶ 321]	Switches monitoring of the global software end positions for the path axes

Minimum velocity

Each NCI group has a minimum path velocity $V_{min} \geq 0.0$. The actual velocity should always exceed this value. User-specified exceptions are: programmed stop at segment transition, path end and override requests which lead to a velocity below the minimum value. A systemic exception is a motion reversal. With the reduction method DEVIATIONANGLE the deflection angle is $\varphi \geq \varphi_h$, in which case the minimum velocity is ignored. V_{min} must be less than the set value for the path velocity (F word) of each segment.

The minimum velocity can be set to a new value $V_{min} \geq 0.0$ in the NC program at any time. The unit is mm/sec.

Classification of the segment transitions

In general, the transition from one segment to the next is not indefinitely smooth. Therefore, it is necessary to reduce the velocity at the transition point in order to avoid dynamic instability. For this purpose, the transitions are geometrically classified and the effective transition velocity - V_{link} - is determined in three categories.

Segments - as geographical objects - are defined here as curves in terms of differential geometry and are parameterized by the arc length.

A segment transition from a segment S_{in} to a segment S_{out} is classified in geometrical terms as type C_k , where k is a natural number (including 0), if each segment has k continuous arc length differentials and the k^{th} derivatives at the transition point correspond.

C0 transitions have a knee-point at the transition point.

C1 transitions appear smooth, but are not smooth in dynamic terms. One example is the straight line-semi circle transition in the stadium: at the transition point there is a step change in acceleration.

C2 transitions (and of course C_k transitions with $k > 2$) are dynamically smooth (jerk restricted).

Reduction method for C2 transitions

As at all transitions, at C2 transitions V_{link} is set to equal the minimum of both set segment velocities: $V_{link} = \min(V_{in}, V_{out})$. There is no further reduction.

Reduction method for C1 transitions

First, V_{link} is set to the lower of the two segment target velocities: $V_{link} = \min(V_{in}, V_{out})$. The geometrically induced absolute step change in acceleration $AccJump$ in the segment transition is calculated depending on the geometry types G_{in} and G_{out} , and the plane selection G_{in} and G_{out} of the segments to be connected, at velocity V_{link} . If this is greater than C1 times the path acceleration/(absolute) deceleration $AccPathReduced$ permissible for the geometries and planes, the velocity V_{link} is reduced until the resulting step change in acceleration is equal to $AccPathReduced$. If this value is less than V_{min} , then V_{min} takes priority.

Note When changing the dynamic parameters, the permissible path acceleration for the geometries and planes and thereby the reaction of the reduction changes automatically.

Interface: [XAE \[▶ 23\]](#) and [interpreter \[▶ 173\]](#)

Reduction modes for C0 transitions

Several reduction methods are available for C0 transitions. The reduction method VELOJUMP reduces the velocity after permitted step changes in velocity for each axis. The reduction method DEVIATIONANGLE reduces the velocity depending on the deflection angle φ (angle between the normalized end tangent T_{in} of the incoming segment S_{in} and the normalized start tangent T_{out} of the outgoing segment S_{out}). The cosine reduction method is a purely geometrical method (see [curve velocity reduction method \[▶ 25\]](#)).

The VELOJUMP method is recommended for mechanically independent axes, while for mechanically coupled axes (the Y axis is attached to the X axis, for example) the DEVIATIONANGLE method is usually recommended.

Reduction method for C0 transitions: VELOJUMP

If $V_{link} = \min(V_{in}, V_{out})$, and for each axis $V_{jump}[i] = CO[i] * \min(A+[i], -A-[i]) * T$ is the permitted absolute step change in velocity for the axis $[i]$, wherein $CO[i]$ is the reduction factor and $A+[i]$, $A-[i]$ are the acceleration/deceleration limits for the axis $[i]$, and T is the cycle time. The VELOJUMP reduction method ensures that the path velocity is reduced at the segment transition V_{link} until the absolute step change in the set axis velocity of axis $[i]$ is at most $V_{jump}[i]$. V_{min} nevertheless has priority: if V_{link} is less than V_{min} , V_{link} is set to V_{min} . In the case of movement reversal with no programmed stop, there will be a jump in axis velocity.

Note When changing the dynamic parameters, the maximum permissible step changes in axis velocity automatically change at the same time.

Reduction method for C0 transitions: DEVIATIONANGLE

Note When changing the dynamic parameters, the reduction factors do not automatically change at the same time.

Changing the parameters for C0 transitions: DEVIATIONANGLE

Table 2: Parameter

Parameter	Meaning and boundary conditions
DEVIATIONANGLE: Reduction factor $C0C0$	Path reduction factor for C0 transitions: $1.0 \geq C0 \geq 0.0$
DEVIATIONANGLE: Critical angle (low) φ_l	Angle from which reduction takes effect: $0 \leq \varphi_l < \varphi_h \leq \pi$
DEVIATIONANGLE: Critical angle (high) φ_h	Angle from which reduction to $v_{link} = 0.0$ takes effect: $0 \leq \varphi_l < \varphi_h \leq \pi$

Interface: [Interpreter](#) [[▶ 173](#)]

Cosine reduction method

See [here](#) [[▶ 25](#)].

Tolerance sphere radius and C2 reduction factor

These parameters are described under the heading [Smoothing of segment transitions](#) [[▶ 125](#)].

Global software limit positions for the path

The 'Global software limit position monitoring for the path' offers two different ways of software position limit monitoring.

Limit position monitoring by the SAF task

This type of end position monitoring is always active if the limit position for the axis has been switched to active (axis parameter). The monitoring is carried out component for component by the SAF task. This means that if the end position is exceeded, the path velocity is instantly set to 0, and the entire interpolation group has an error.

This type of monitoring is activated through the axes parameters, and **not** by means of the group parameters described here.

Software limit positions on the path

To prevent the path velocity being set to 0 immediately when a violation of the software end positions is encountered, the function 'Global software end position monitoring of the path' must be enabled. If this is active, the movement stops at the NC block in which the end positions were violated. The velocity is reduced via a ramp.

- So that the monitoring is only executed for the desired path axes, the software limit positions for the axis components must be selected (axis parameters).
- The monitoring is carried out for the standard geometry segments. These include
 - Straight line
 - Circle
 - Helix
- Curves with splines are not monitored. The set values associated with the splines are always within the tolerance sphere. Otherwise the limit position monitoring will make use of the SAF task.
- Because meaningful and generally applicable monitoring of the end positions can only be carried out at the NC program's run-time (before lookahead) it is possible that the path axes will move as far as (but not including) the NC block in which the limit positions are exceeded.

- If for some reason the axes are located outside the software limit positions it is possible to move back into the correct region in a straight line.

Parameterization:

XAE: [Group parameters](#) [► 23]

8.4.1 Path override (interpreter override types)

The path override is a velocity override. This means that changing the override creates a new velocity, but does not affect the ramps (acceleration or jerk). The used override types only differ in terms of reference velocity.

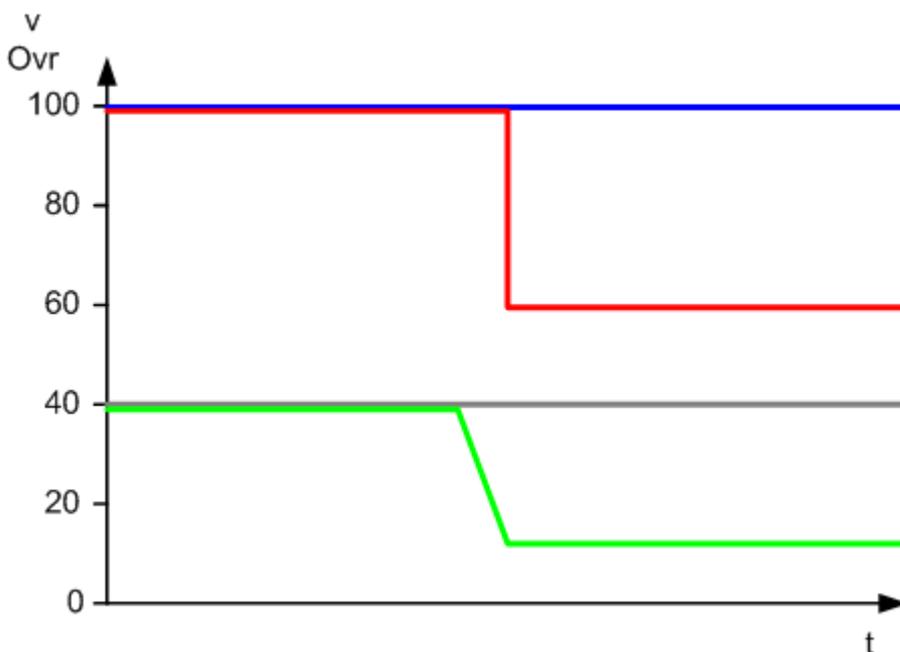
The parameterization takes place in the interpolation channel under the [group parameters](#) [► 24].

Option 'Reduced' - based on the reduced velocity (default)

Because of the relevant dynamic parameters (braking distance, acceleration etc.) it is not possible for the programmed velocity (the blue line) be achieved in every segment. For this reason a velocity, possibly reduced, (the red line) is calculated for each geometric segment. In the standard case, the override is made with reference to this segment velocity.

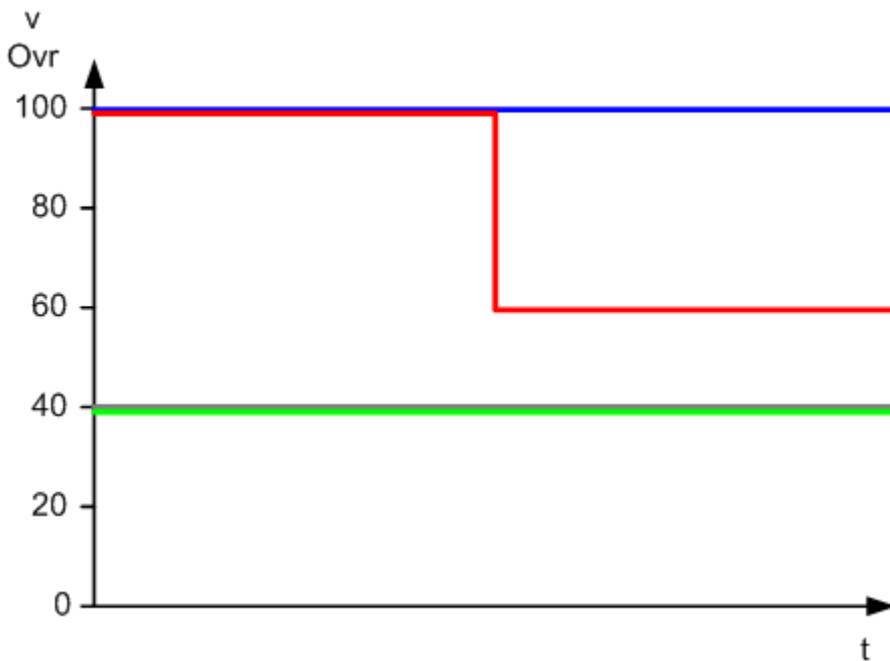
The advantage of this override type is that if override values are small the machine operates with an approximately linear reduction in velocity, and this is therefore the correct setting for most applications.

$$v_{\text{res}} = v_{\text{max}} * \text{Override}$$



Option 'Original' - based on the programmed path velocity

The override value is based on the velocity programmed by the user. The maximum segment velocity only has a limiting effect.



Selection 'Reduced [0 ... >100%]' - based on internally reduced velocity with the option to specify a value greater than 100%

The override type behaves like 'Reduced' [▶ 322]. With this override type it is possible to travel along the path more quickly than programmed in the G-Code. There is no limitation to 120%, for example. The maximum possible path velocity is limited by the maximum velocities of the axis components (G0 velocity) and their dynamics.

If limitation to a particular value, e.g. 120%, is required, this can be set in the PLC project.

8.5 Cyclic Channel Interface

The channel interface is responsible for the cyclic data exchange between the PLC and the NCI.

From the NCI to the PLC (160 bytes)

```

TYPE NCTOPLC_NCICHANNEL_REF :
STRUCT
BlockNo           : UDINT;
FastMFuncMask     : ARRAY [1..5] OF DWORD;
HskMFuncNo       : UINT;
HskMFuncReq      : WORD;
HFuncValue       : UDINT;
SpindleRpm       : UINT;
Tool             : UINT;
ChnState         : NCTOPLC_NCICHANNEL_REF_CHN_STATE;
IntParams        : ARRAY [0..3] OF UDINT;
DoubleParams     : ARRAY [0..3] OF LREAL;
PathVelo         : LREAL;
LoadedProg       : UDINT;
ItpMode          : WORD;
ItpState         : UINT;
ErrorCode        : UDINT;
ChnId           : UINT;
GrpId           : UINT;
ItfVersion       : UINT;
_reserved1       : UINT;
ChnOperationState : UDINT;
McsAxisIDs      : ARRAY [0..7] OF USINT;
AcsAxisIDs      : ARRAY [0..7] OF USINT;
_reserved2       : ARRAY [1..24] OF USINT;
END_STRUCT
END_TYPE

```

Variable name	Data type	Description
BlockNo	UDINT	block number
FastMFuncMask	ARRAY OF DWORD	Bit mask for evaluation of the fast M-functions [▶ 161]
HskMFuncNo	UINT	Number of synchronous M-function present (M-function with handshake)
HskMFuncReq	WORD	Flag indicating that a synchronous M-function is present 0: no synchronous M-function is present 1: a synchronous M-function is present
HFuncValue	DINT	Value of the auxiliary function
SpindleRpm	WORD	Spindle rotation speed
Tool	WORD	Tool number
ChnState	NCTOPLC_NCICHANNEL_REF_CHN_STATE	DWORD with status information for the channel (see status information for the channel (ChnState) [▶ 324])
IntParams	ARRAY [0..3] OF UDINT	Data of the freely configurable channel interface (see ItpSetCyclicUDintOffsets [▶ 236])
DoubleParams	ARRAY [0..3] OF LREAL	Data of the freely configurable channel interface (see ItpSetCyclicLrealOffsets [▶ 235])
PathVelo	LREAL	Current path set velocity
LoadedProg	UDINT	Name of the currently executed NC program. If the name is not a UDINT, this value is 0.
ItpMode	WORD	Bit mask that indicates execution in interpreter mode.
ItpState	UINT	Status [▶ 15] of the interpreter
ErrorCode	UDINT	Error code of the interpreter channel
ChnId	UINT	Channel ID
GrpId	UINT	group ID
ItpVersion	UINT	Version of this cyclic channel interface
ChnOperationState	UDINT	Channel state for a channel of the kinematic transformation; has no purpose for an interpolation channel.
McsAxisIDs	ARRAY [0..7] OF USINT	IDs of the MCS axes for a kinematic transformation channel; has no purpose for an interpolation channel.
AcsAxisIDs	ARRAY [0..7] OF USINT	IDs of the ACS axes for a kinematic transformation channel; has no purpose for an interpolation channel.

Channel status information (ChnState)

In the XAE the channel status information can only be read with a plain text name, from the PLC only via the bit number.

Name	Bit number (zero based)	Description
blsInterpolationChannel	0	Indicates that the linked channel is an interpolation channel.
blsKinematicChannel	1	Indicates that the structure is linked to a channel for the kinematic transformation.
blsEStopRequested	8	Indicates that an ItpEStop was called, without checking whether the axes are already at standstill.
blsFeedFromBackupList	10	For retracing the current entries from the interpreter backup list are sent.
blsMovingBackward	11	Indicates that the current motion is a reversing motion.
bRetraceStartPosReached	12	Indicates that the program start was reached during reversing.

From PLC to NCI (128 bytes)

```

TYPE PLCTONC_NCICHANNEL_REF :
STRUCT
SkipLine      : WORD; (* Mask to skip lines *)
ItpMode       : WORD;
MFuncGranted  : WORD; (* granted signal of the M-function *)
_reserved1    : UINT;
ChnAxesOvr   : UDINT; (* Channel override in percent * 100 *)
ChnSpindleOvr : UDINT;
_reserved2    : ARRAY [1..112] OF USINT;
END_STRUCT
END_TYPE
    
```

Variable name	Data type	Description
SkipLine	WORD	Bit mask with which block skipping [► 123] of the NCI is parameterized from the PLC
ItpMode	WORD	Bit mask with which the interpreter execution mode can be altered. This is, for instance, required if the interpreter is to operate in single block [► 129] mode.
MFuncGranted	WORD	Flag with which an M-function of type 'Handshake' is confirmed. 0: Not acknowledged 1: Acknowledgement
ChnAxesOvr	UDINT	Channel override for the axes from 0...1000000 (corresponds to 0 - 100%)
ChnSpindleOvr	UDINT	Channel override for the spindle between 0 and 1000000 (corresponds to 0 - 100%); currently not supported.

More Information:
www.beckhoff.com/tf5100

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

