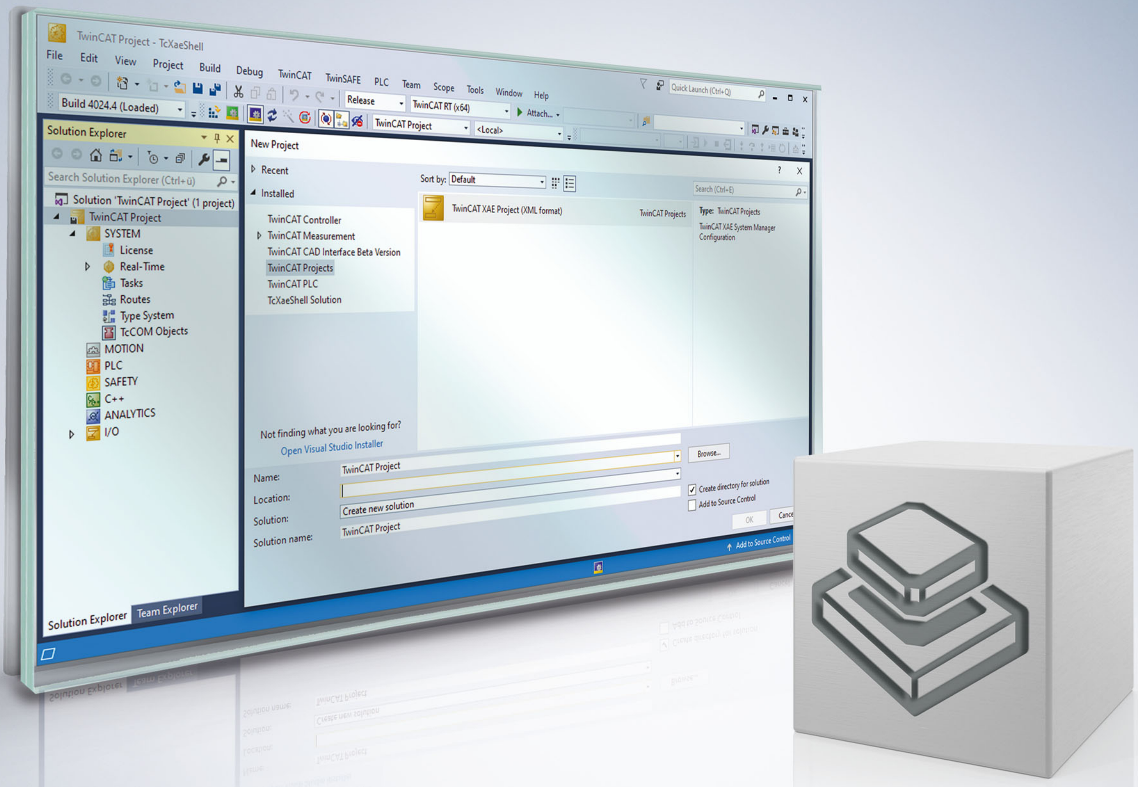
BECKHOFF New Automation Technology

Manual | EN

TF5430

TwinCAT 3 | Planar Motion



2024-03-11 | Version: 1.4.1

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all 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 notice.
No claims to modify 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 and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT

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

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

ℹ This information includes, for example:
recommendations for action, assistance or further information on the product.

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.

Table of contents

2 Overview of the new functions

From version V3.2.60:

- New: The surface that the movers travel on is divided into Planar parts that can be moved during runtime. Movers can cross boundaries between connected or adjacent Planar parts.
- Requires TwinCAT V3.1.4024.40 or higher

From version V3.1.10.63:

- Requires TwinCAT V3.1.4024.24 or higher

From version V3.1.10.51:

- New: AdoptTrackOrientation rotates the mover on the track to align it with the track orientation. This changes the C coordinate mode from independent to dependent.
- Advanced: MoveC now always works for standing movers on the track. This may change the C coordinate mode from dependent to independent.

From version V3.1.10.44:

- New: GearInPosOnTrack and GearInPosOnTrackWithMasterMover commands for coupling a Planar mover to a master axis or a master mover, respectively
- Advanced: Parameters of the Planar track TcCOM module
- Requires TwinCAT V3.1.4024.17 or higher

From version V3.1.10.30:

- New: CRotation command mode (360° rotation) with modulo positioning
- New: Constraints as a new variant to limit the dynamics of the motion commands
- Advanced: Parameters for modulo positioning on "Closed Loop" tracks

From version V3.1.10.11:

- First version of Planar Motion released
- Requires TwinCAT V3.1.4024.12 or higher

3 Introduction

The TwinCAT 3 Planar Motion software package TF5430 is installed together with the software package TF5400.

Target system

Windows 7/8/10 (only 64-bit)

TwinCAT 3 Planar Motion Base

The TF5430 TwinCAT 3 Planar Motion software combines a wide range of functionalities for controlling XPlanar movers and enables efficient and intelligent implementation of individual XPlanar applications. TF5430 TwinCAT 3 Planar Motion is part of TF5890 TwinCAT 3 XPlanar. All associated function blocks are included in the library Tc3_Mc3PlanarMotion, which is to be used in combination with the library Tc3_Physics.

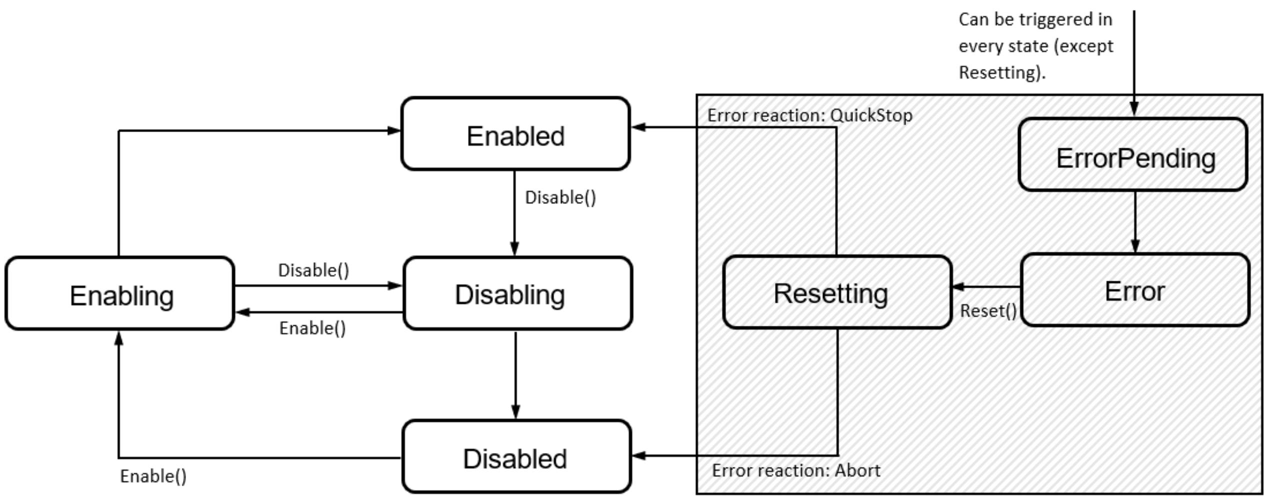
Additional licensing requirements

TF5430 TwinCAT 3 Planar Motion requires the TC1250 license.

4 States and modes

4.1 Planar objects state diagram

The Planar State Machine is used by the Planar mover, the Planar track and the Planar group. All of these components can be in the *seven* planar states: Enabling, Enabled, Disabling, Disabled, Resetting, ErrorPending, Error.



The error reaction depends on how serious the error is. For minor errors, the normal error reaction is a QuickStop. The user can also force the error reaction Abort by Disable() in this case. The command must be sent in one of the three states: ErrorPending, Error or Resetting.

Enabling

In the Enabling state, the Enable command is executed. At the end of this command, the component is in the Enabled state. In the Enabling state, a Disable command can be sent that cancels the Enable command and causes the state to change to Disabling.

Enabled

In the Enabled state the component is fully functional and can be used by the user. In this state a Disable command can be sent. The state then switches to Disabling.

Disabling

In the Disabling state the Disable command is executed. At the end of this command, the component is in the Disabled state. In the Disabling state, an Enable command can be sent that cancels the Disable command and causes the state to change to Enabling.

Disabled

After the system is booted the components are in the Disabled state. They can be placed in the Enabling state using an Enable command. The components are not functional in the Disabled state.

Resetting

The component is in the process of rectifying the error. Depending on the error reaction it is then in the Enabled or Disabled state.

ErrorPending

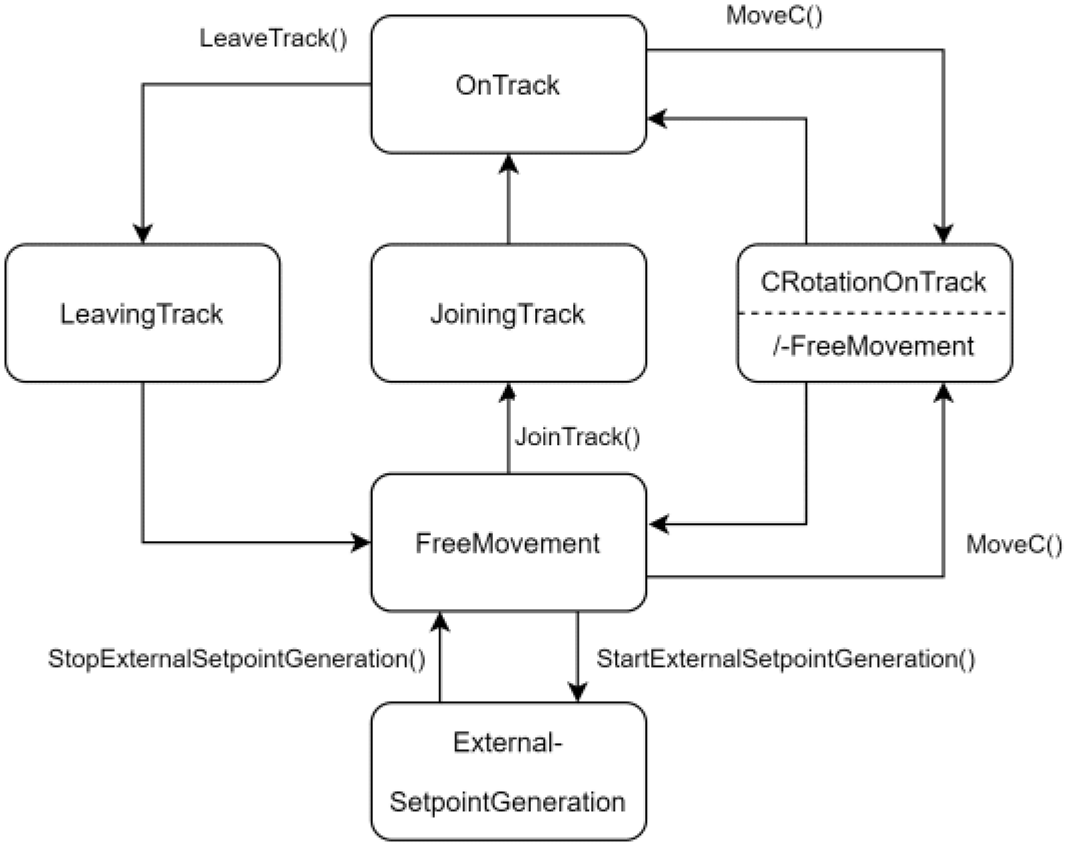
When an error occurs the component reaches the ErrorPending state from all other states except the Resetting state. Once the error has been processed correctly, the state switches to Error.

Error

The Error state means that an error has occurred and the component can now be placed in the Resetting state using the Reset command in order to correct the error.

4.2 Planar mover command diagram

The Planar mover has six different command modes that indicate what type of command the mover executes: OnTrack, LeavingTrack, JoiningTrack, ExternalSetpointGeneration and CRotationFreeMovement/-OnTrack (from Version V3.1.10.51). In all modes except ExternalSetpointGeneration mode, collision avoidance is active for the mover when it is in a group.



OnTrack

In the OnTrack mode the mover joins a track and can be moved on it (MoveOnTrack). The mover can also leave the track again (LeaveTrack), which changes the mode to LeavingTrack. MoveC commands cause a change to CRotationOnTrack mode if necessary.

LeavingTrack

In LeavingTrack mode the mover does not accept any further commands. The mode is quit automatically when the mover has ended the LeaveTrack command. The mover is then in FreeMovement mode.

JoiningTrack

In JoiningTrack mode the mover does not accept any further commands. The mode is quit automatically when the mover has ended the JoinTrack command. The mover is then in the OnTrack mode.

FreeMovement

After enabling the mover, it is automatically in this command mode, unless the mover is twisted too much. Then it is in CRotationFreeMovement mode. The mover can be moved freely with MoveToPosition commands. If the user starts the external setpoint generation via a command, the mode switches to ExternalSetpointGeneration. JoinTrack commands are also possible that change the mode to JoiningTrack. MoveC commands may cause a change to the CRotationFreeMovement mode.

CRotationFreeMovement/-OnTrack

This mode is started by the MoveC command if the C-movement that is in progress does not take place entirely within a C-position window. The windows are defined by the position limits of the C-axis of the mover and exist 4 times each rotated by 90° (the 90° rotation results from the mover symmetry: e.g. limits +-15° -> window 1. [-15°,+15°], 2. [75°,105°], 3. [165°,195°], 4. [255°,285°]). Depending on whether you were previously in FreeMovement or OnTrack mode, the mode will be CRotationFreeMovement or CRotationOnTrack accordingly. The mode is finished when the C-movement is completed and the end position is within one of the four windows. The mover then automatically returns to the previous mode. The mover thus changes from CRotationFreeMovement to *FreeMovement* or from CRotationOnTrack to *OnTrack*. Otherwise, it remains in CrotationFreeMovement/-OnTrack mode. In both CRotation modes, the X and Y-axes of the mover cannot be moved. If the mover already has an orientation outside the 4 windows when it starts up, it is immediately in CRotationFreeMovement mode instead of FreeMovement.

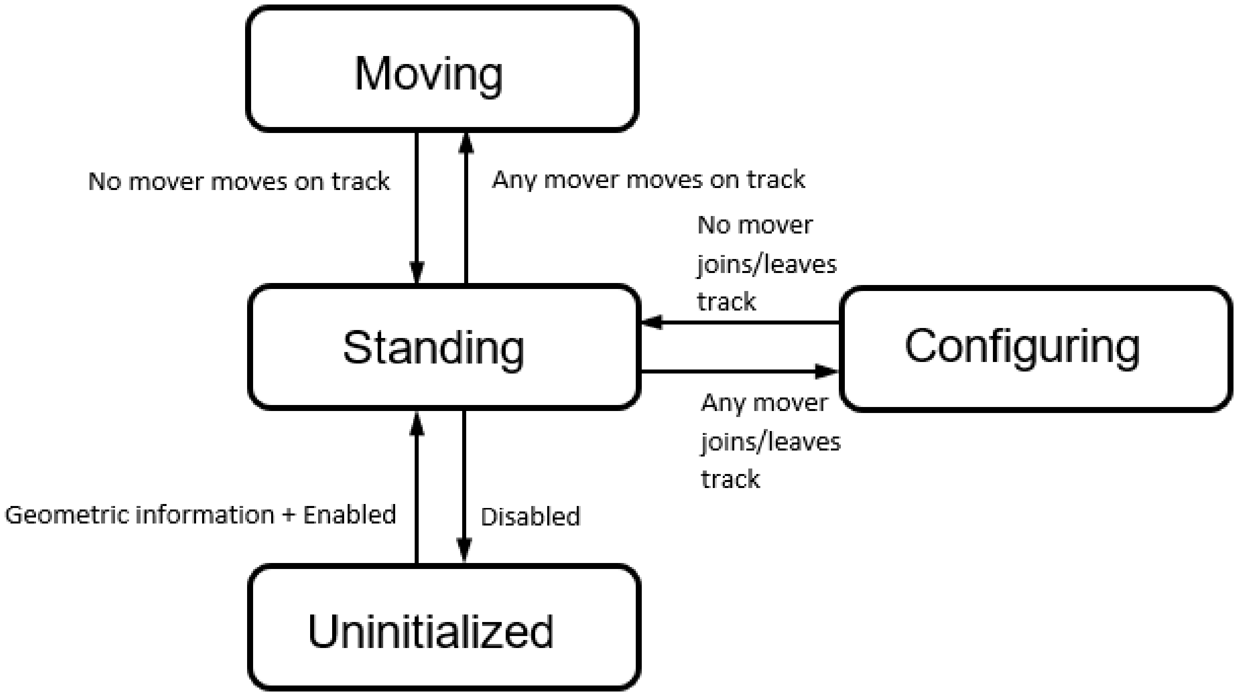
ExternalSetpointGeneration

In ExternalSetpointGeneration mode, the mover executes a corresponding command. This mode begins (or ends) with the beginning (or end) of the corresponding command. In the ExternalSetpointGeneration mode, the mover follows the external setpoints that the user provides cyclically.

External setpoint generation can be used in conjunction with the other modes. In this case, the external setpoints are simply added as relative offsets from the setpoints of the other modes. However, the mover is then not in ExternalSetpointGeneration mode.

4.3 Planar track operation modes

The Planar track has four different operation modes that indicate whether and how the track performs or can perform its function as a "Street for Movers": Moving, Standing, Configuring and Uninitialized.



Moving

In Moving mode, one or more movers are about to move on the track (MoveOnTrack). The first mover to start a movement on the track in Standing mode automatically changes the mode from Standing to Moving. Accordingly, the last mover that completes its movement changes the mode back to Standing. No mover is allowed to execute a JoinTrack or LeaveTrack command while the track is in Moving mode. If the track is in a Planar group, it blocks its surface.

Standing

In Standing mode the track is usable by movers. All movers on the track are standing and waiting for travel commands. JoinTrack, LeaveTrack and MoveOnTrack commands are allowed for the movers in this mode. Each of these commands ends the Standing mode of the track. If the track is in a Planar group, it does not block its surface.

Configuring

In Configuring mode, one or more movers are about to leave the track (LeaveTrack) or join the track (JoinTrack). The first mover to leave (or join) the track in Standing mode automatically changes the mode from Standing to Configuring. Accordingly, the last mover to complete leaving or joining changes the mode back to Standing. No mover is allowed to execute a MoveOnTrack command while the track is in Configuring mode. If the track is in a Planar group, it does not block its surface.

Uninitialized

The track is not usable by movers in the Uninitialized mode. It does not have a finished geometric description yet. When the user creates and enables this geometric description, the track switches to Standing mode.

5 Parts

5.1 Parts and coordinate systems

From version V3.2.60: The Part feature, which is the subject of this section, is available.

The surface that the XPlanar movers move on is made up of stators in the form of squares with a side length of 240 mm. The entire base is divided into one or more parts consisting of one or more stators. A stator can only belong to one part, i.e. the parts do not overlap. The total number of stators of a part must be assembled in a contiguous surface.

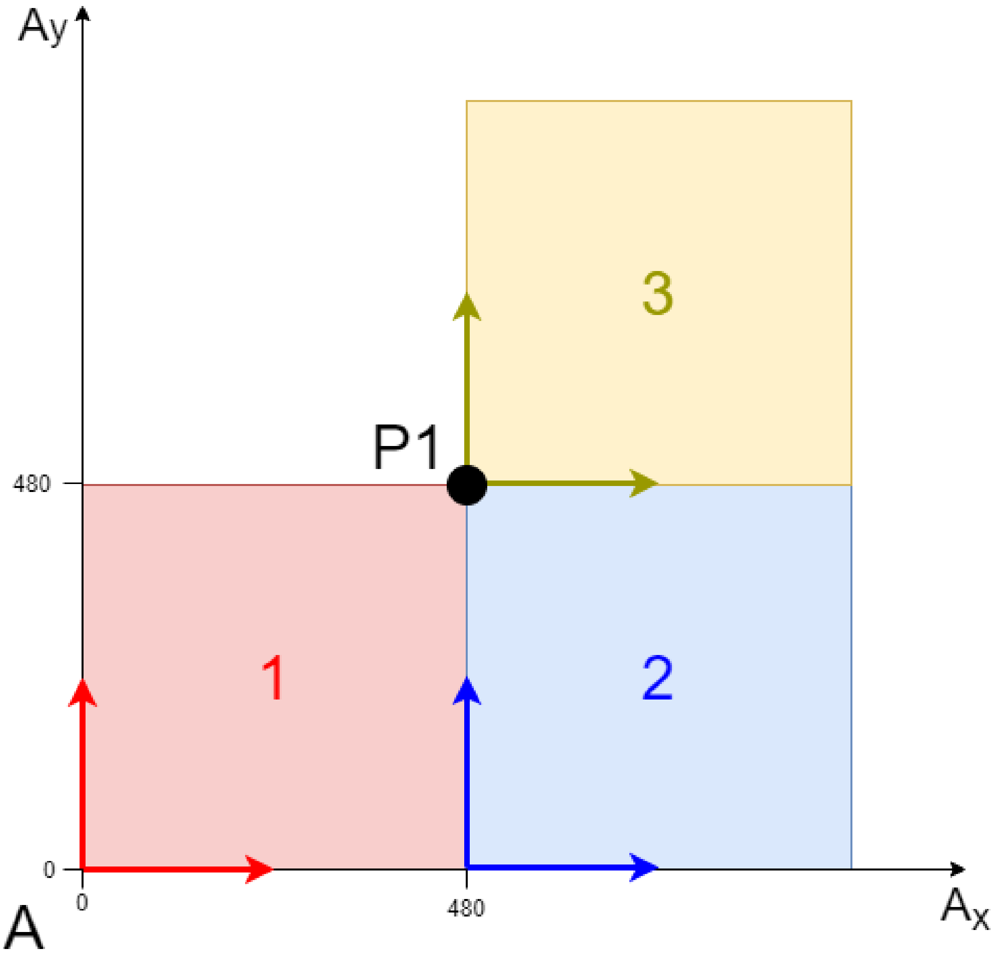
This geometric configuration of the XPlanar system is usually static; it does not change during system runtime. To create a dynamic configuration, the user must move or relocate individual parts by assigning them more than one position and activating them at runtime.

In general, an issue arises: the positions are no longer unique or there is no absolute coordinate system. The user can define several (2D) coordinate systems and place his parts in these.

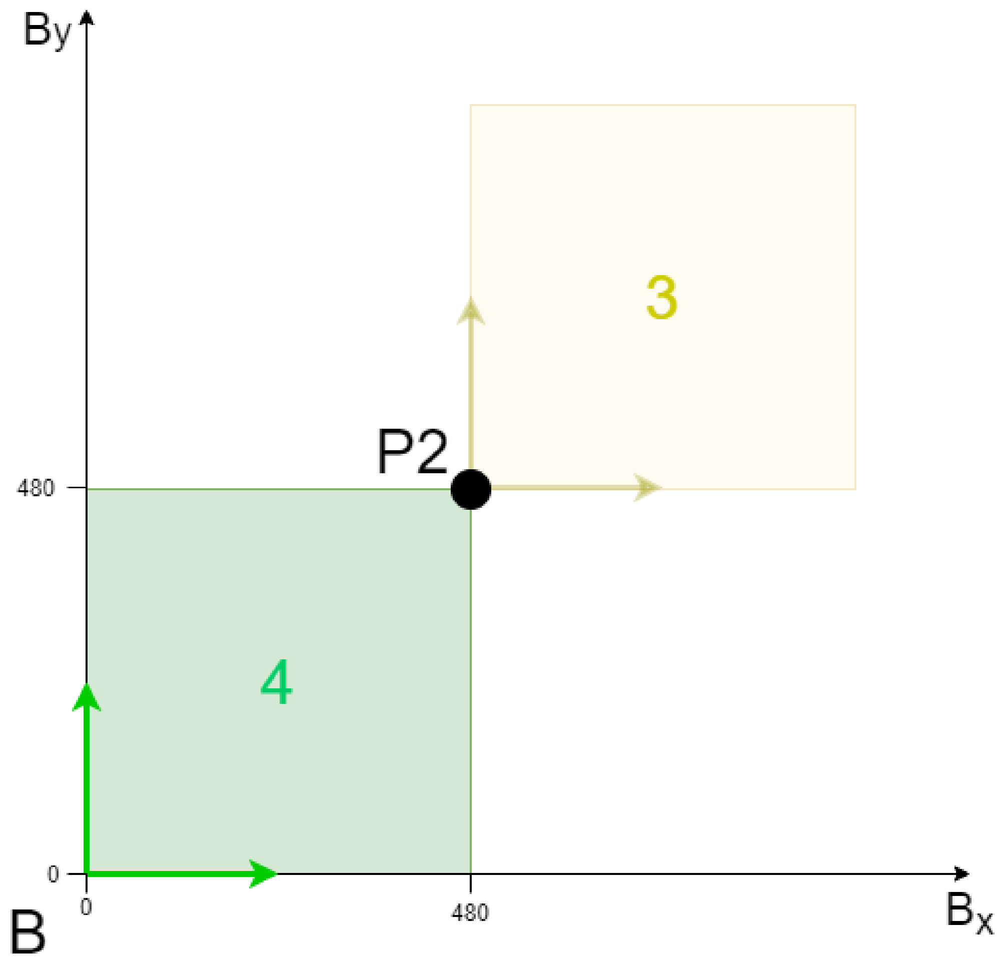
A 2D position (e.g. of a mover or track) is now incomplete without an indication of *which* coordinate system it is located in.

Example

There are four parts, 1-4, and two coordinate systems, A and B. Part 3 is either in coordinate system A or B. The other parts are permanently assigned to a coordinate system. The position P1 is located in the coordinate system A at the position X=480 and y=480 (P1=(480,480,A) for short).



The position P2 is located in the coordinate system B at the position X=480 and y=480 (P2=(480,480,B) for short).



Without specifying the coordinate system A, the position P1 would not be distinguishable from the position P2 in coordinate system B (480,480,B). However, both positions are different and are not even in the same coordinate system, i.e. there is no geometric connection between them (e.g. a mover cannot travel from P1 to P2).

In addition to coordinate systems A and B, there are also local part coordinate systems for each of the parts 1-4. These coordinate systems each have their origin in the bottom left-hand corner of their part. This means that P1 = (0,0,3) = (0,480,2) = (480,480,1) in the part coordinate systems of parts 1-3.

This is only true for part 3 as long as it lies in coordinate system A. The transparent representation of part 3 in coordinate system B shows that part 3 can change its position from coordinate system A to coordinate system B. The position of a part is identical to the origin of its coordinate system. Part 3 would therefore be moved from position P1 (in coordinate system A) to position P2, and now P1 can no longer be specified in the coordinate system of part 3.

Instead, the position P2 = (0,0,3) = (480,480,4) can be specified in the coordinate system of part 3 (and 4). Overall, you can see that positions in part coordinate systems *are not static* if the part is moved with its coordinate system.

5.2 Configuration

From version V3.2.60: The Part feature, which is the subject of this section, is available.

The configuration of the stators of a part (i.e. the geometric extent) and the part positions in the various coordinate systems takes place in the XPlanar driver and not in the **MC Configuration**. Therefore we will not describe how to create this configuration here.

However, this information is important for both the **MC Configuration** and for PLC control, so it is read from the XPlanar driver by the Planar environment when it is activated and distributed from there to all Planar objects within the **MC Configuration**. In the PLC, a separate MC_PlanarPart [▶ 158] object is used to control the part position and states.

The item objects, PositionXYC and PositionXY, are extended to specify items uniquely. They now contain the ID of the coordinate system (ReferenceId) in addition to the coordinate values. As the positions are no longer unique, the position of the mover can now be specified in two ways:

1. As the position in the higher-level coordinate system (that the part that the mover is on is located in).
2. As a position in the part coordinate system. The position in the higher-level coordinate system is specified in the cyclical interface. In addition, its position in the part coordinate system can be queried using the method GetPositionOnCurrentPart [▶ 157].

Tracks can be connected to various other tracks depending on the part position. These connections can be made using two methods, StartFromTrackAdvanced [▶ 166] and EndAtTrackAdvanced [▶ 167]. For external setpoint generation, the coordinate system in which the external setpoint is located must also be specified. This is done using the method SetExternalSetpointReferenceId.

These functionalities of the **MC Configuration** are described in detail below.

5.3 Positions with ReferenceId

From version V3.2.60: The Part feature, which is the subject of this section, is available.

With the introduction of parts and coordinate systems, positions no longer make sense without specifying the reference system in which they are located. For this purpose, the "ReferenceId" property has been added for the PositionXYC and PositionXY objects. You now also save the ID of the reference system.

These two position objects, PositionXYC and PositionXY, can (and should) now always be used with the explicit ID of the reference system. Stating a specific reference system is safer, as this makes it explicitly clear which system is meant.

Not specifying a reference system or specifying the "zero" reference system is only permitted in exceptional cases if there is only one static coordinate system in which all parts have a fixed position. The ID "zero" is then converted internally into the ID of the sole coordinate system. In all other cases, the reference system "zero" is rejected. Non-valid reference systems (invalid object ID of a part/coordinate system) other than "zero" are always rejected.

6 Planar Motion components

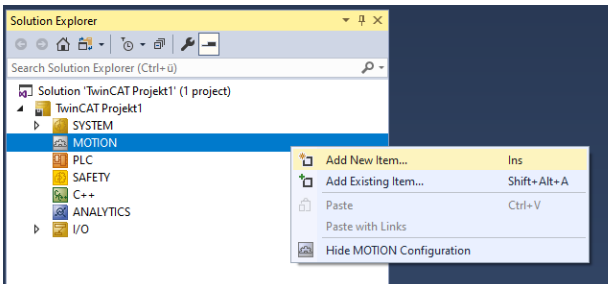
6.1 Planar mover

The Planar mover is a software object that represents an XPlanar mover. It summarizes the state of the real mover (position, velocity, etc.) for the user. In addition, the user has the possibility to influence or control the state of the real mover via the Planar mover.

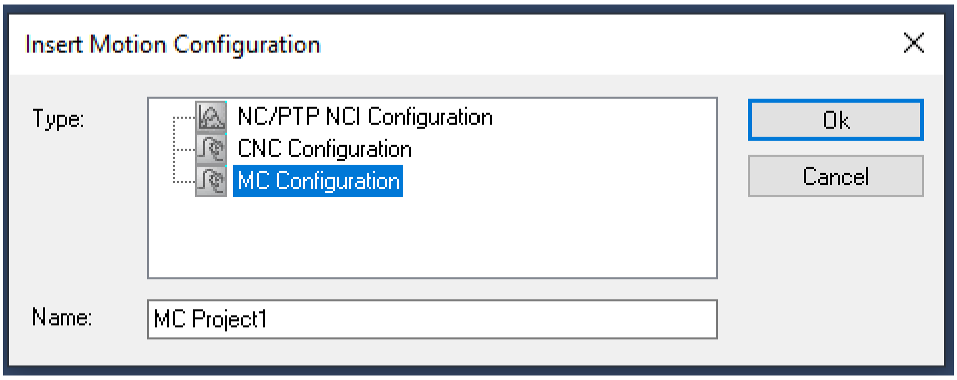
6.1.1 Configuration

✓ In order to create a Planar mover, an **MC Configuration** must first be created.

1. Select **MOTION** > **Add New Item…**.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Projekt1' (1 project)
TwinCAT Projekt1
SYSTEM
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
Add New Item... Ins
Add Existing Item... Shift+Alt+A
Paste Ctrl+V
Paste with Links
Hide MOTION Configuration

2. In the following dialog box, select **MC Configuration** and confirm with **OK**.

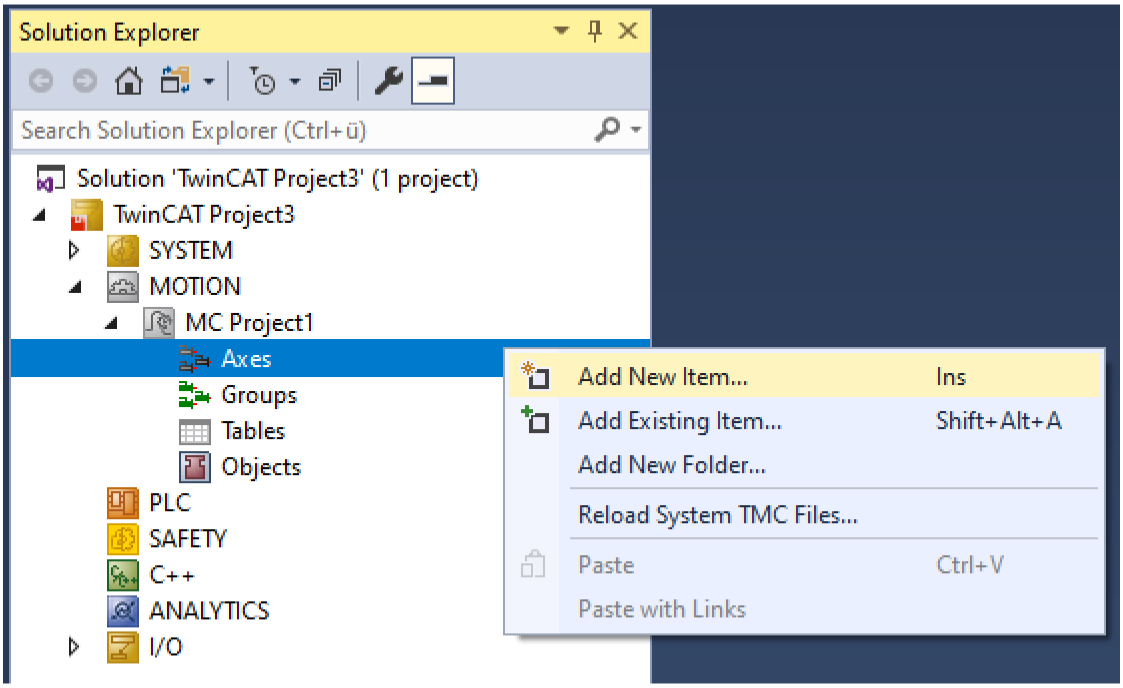

Insert Motion Configuration

Type: NC/PTP NCI Configuration, CNC Configuration, MC Configuration

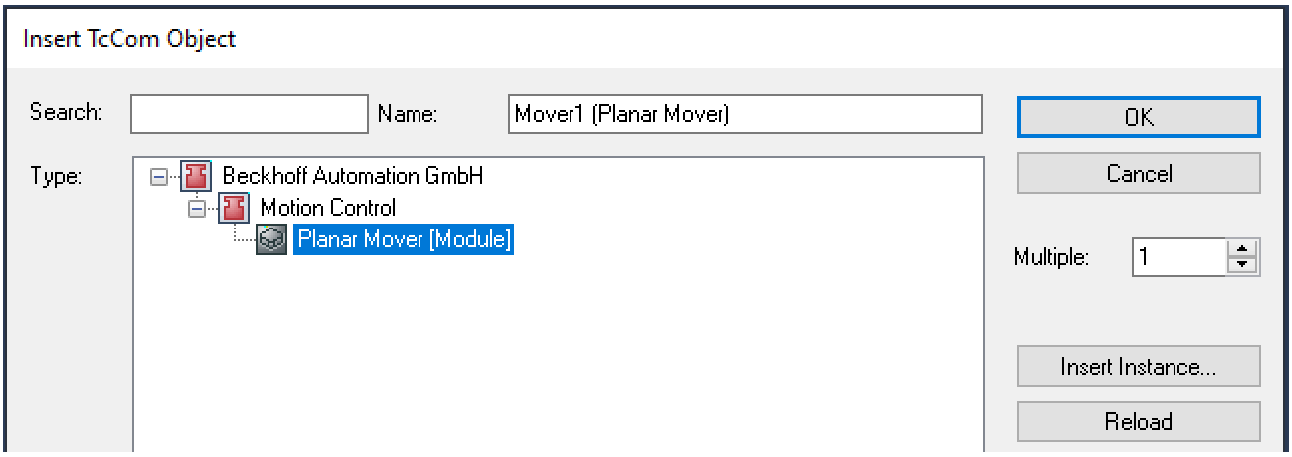
Name: MC Project1

⇨ You have created an MC Project.

3. Select **MC Project** > **Axes** > **Add New Item…**.



4. In the following dialog box, create one (or more) Planar movers and confirm with **OK**.



Insert TcCom Object

Search:

Name: Mover1 (Planar Mover)

Type:
- Beckhoff Automation GmbH
  - Motion Control
    - Planar Mover [Module]

Multiple: 1

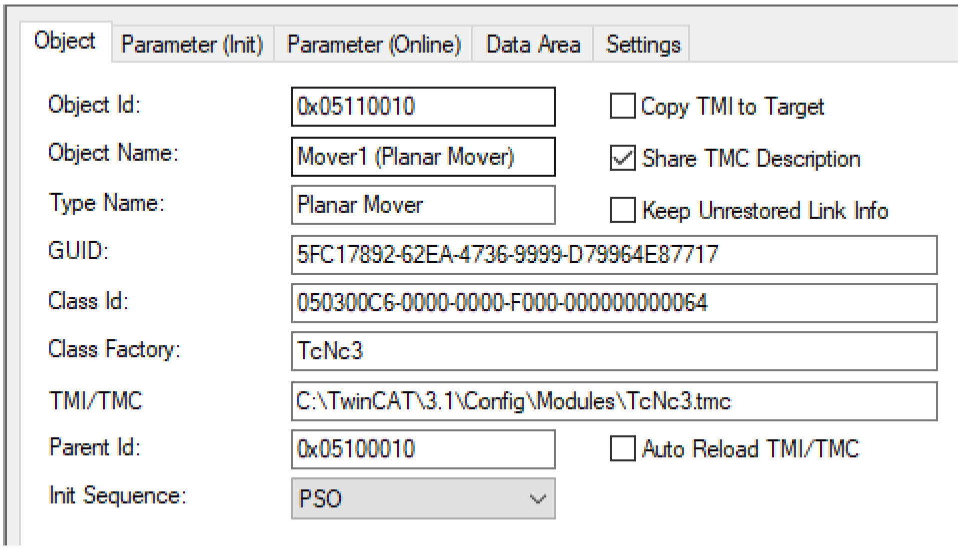
⇨ The Planar mover is now created and can be parameterized.

Open detailed description

- Select the Planar mover in the tree and double-click it.

Purposes of the individual tabs

Object: General information (name, type, ID and so on) is shown here.



Object Id: 0x05110010

Object Name: Mover1 (Planar Mover)

Type Name: Planar Mover

GUID: 5FC17892-62EA-4736-9999-D79964E87717

Class Id: 050300C6-0000-0000-F000-000000000064

Class Factory: TcNc3

TMI/TMC: C:\TwinCAT\3.1\Config\Modules\TcNc3.tmc

Parent Id: 0x05100010

Init Sequence: PSO

□ Copy TMI to Target

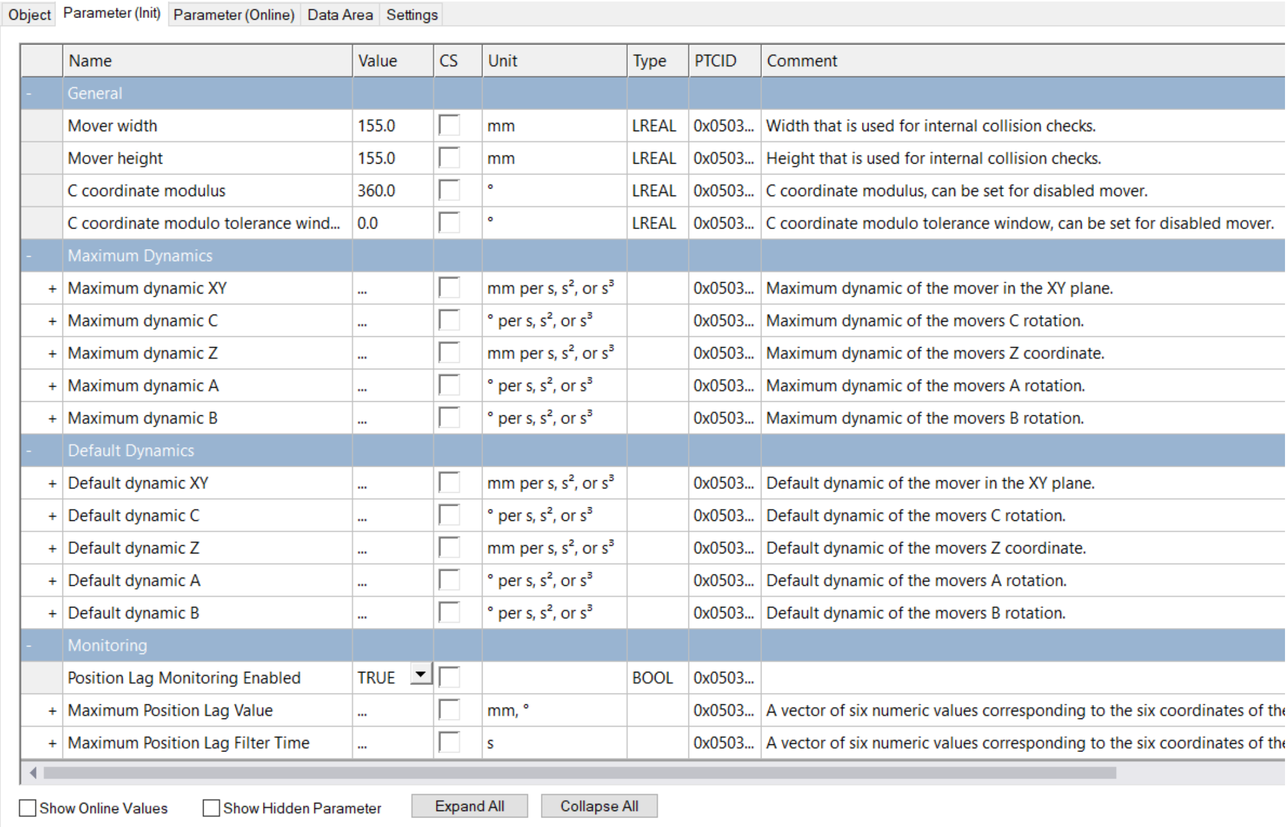
☑ Share TMC Description

□ Keep Unrestored Link Info

□ Auto Reload TMI/TMC

Parameter (Init): Specifies initial parameters that the user can change in order to affect the behavior of the mover.

Parameter (Init) should be put into simulation mode (`TRUE`) before parameterizing if no hardware driver is linked. The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.



	Name	Value	CS	Unit	Type	PTCID	Comment
-	General						
	Mover width	155.0	□	mm	LREAL	0x0503...	Width that is used for internal collision checks.
	Mover height	155.0	□	mm	LREAL	0x0503...	Height that is used for internal collision checks.
	C coordinate modulus	360.0	□	°	LREAL	0x0503...	C coordinate modulus, can be set for disabled mover.
	C coordinate modulo tolerance wind...	0.0	□	°	LREAL	0x0503...	C coordinate modulo tolerance window, can be set for disabled mover.
-	Maximum Dynamics						
+	Maximum dynamic XY	...	□	mm per s, s^2, or s^3		0x0503...	Maximum dynamic of the mover in the XY plane.
+	Maximum dynamic C	...	□	° per s, s^2, or s^3		0x0503...	Maximum dynamic of the movers C rotation.
+	Maximum dynamic Z	...	□	mm per s, s^2, or s^3		0x0503...	Maximum dynamic of the movers Z coordinate.
+	Maximum dynamic A	...	□	° per s, s^2, or s^3		0x0503...	Maximum dynamic of the movers A rotation.
+	Maximum dynamic B	...	□	° per s, s^2, or s^3		0x0503...	Maximum dynamic of the movers B rotation.
-	Default Dynamics						
+	Default dynamic XY	...	□	mm per s, s^2, or s^3		0x0503...	Default dynamic of the mover in the XY plane.
+	Default dynamic C	...	□	° per s, s^2, or s^3		0x0503...	Default dynamic of the movers C rotation.
+	Default dynamic Z	...	□	mm per s, s^2, or s^3		0x0503...	Default dynamic of the movers Z coordinate.
+	Default dynamic A	...	□	° per s, s^2, or s^3		0x0503...	Default dynamic of the movers A rotation.
+	Default dynamic B	...	□	° per s, s^2, or s^3		0x0503...	Default dynamic of the movers B rotation.
-	Monitoring						
	Position Lag Monitoring Enabled	TRUE	□		BOOL	0x0503...	
+	Maximum Position Lag Value	...	□	mm, °		0x0503...	A vector of six numeric values corresponding to the six coordinates of the
+	Maximum Position Lag Filter Time	...	□	s		0x0503...	A vector of six numeric values corresponding to the six coordinates of the

□ Show Online Values □ Show Hidden Parameter

The initial parameters are initially set so that the Planar mover (ready linked) can be moved with the hardware. If the user wants to move without hardware, the "Simulation Mode" parameter must be set to TRUE. In simulation mode, the "Initial Position" and "PartOID" parameters (**from version V3.2.60**) should be set. If the real mover does not have standard dimensions, the "Mover width" and "Mover height" parameters must be adjusted.

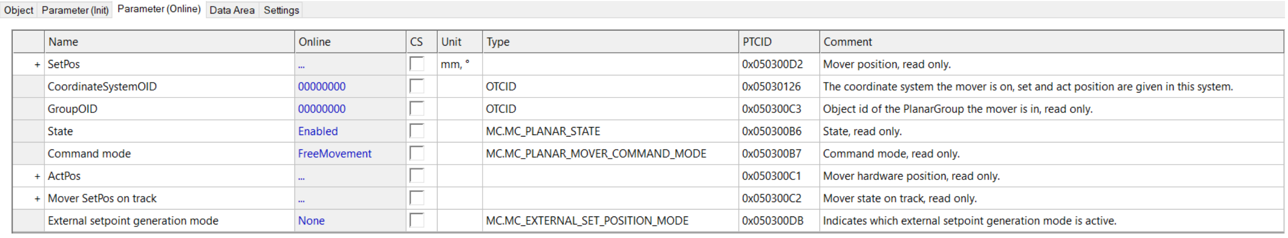
From version V3.1.10.30: The hidden "Minimum/Maximum Position" parameters are used to define when the mover switches to the CRotation command mode for the C-axis. For all target positions of C-movements, the "C coordinate modulus" and "C coordinate modulo tolerance window" parameters (the latter for modulo positioning) define the conversion to the absolute target position. For details see Modulo positioning.

From version V3.1.10.51: AdoptTrackOrientation is also a C-movement and is accordingly influenced by "C coordinate modulus" and "C coordinate modulo tolerance window". For details, see AdoptTrackOrientation [▶ 62].

From version V3.2.60: The "PartOID" parameter specifies the part that the "Initial Position" is located on for the simulation mode. The "PartOID", "Simulation Mode", and "Initial Position" parameters are all hidden and in their own "Simulation" grouping.

Other parameters are the "Maximum Dynamic(s)" and the "Default Dynamic(s)". In addition, there are "monitoring" parameters that activate or parameterize position monitoring of the real mover.

Parameter (Online): Shows the state of the mover during the runtime of the object. The current preset position ("SetPos") and real position ("ActPos") are displayed along with the state information.

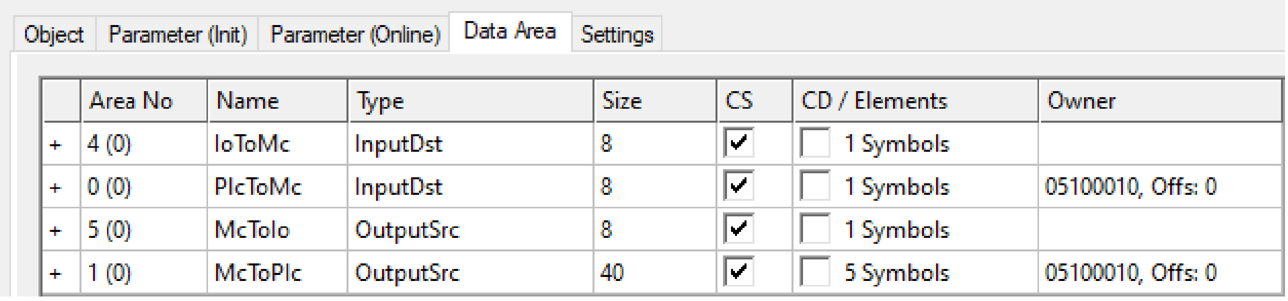


	Name	Online	CS	Unit	Type	PTCID	Comment
+	SetPos	...	□	mm, °		0x050300D2	Mover position, read only.
	CoordinateSystemOID	00000000	□		OTCID	0x05030126	The coordinate system the mover is on, set and act position are given in this system.
	GroupOID	00000000	□		OTCID	0x050300C3	Object id of the PlanarGroup the mover is in, read only.
	State	Enabled	□		MC.MC_PLANAR_STATE	0x050300B6	State, read only.
	Command mode	FreeMovement	□		MC.MC_PLANAR_MOVER_COMMAND_MODE	0x050300B7	Command mode, read only.
+	ActPos	...	□			0x050300C1	Mover hardware position, read only.
+	Mover SetPos on track	...	□			0x050300C2	Mover state on track, read only.
	External setpoint generation mode	None	□		MC.MC_EXTERNAL_SET_POSITION_MODE	0x050300DB	Indicates which external setpoint generation mode is active.

From version V3.1.10.30: The parameter "External setpoint generation" indicates whether the mover follows (absolute or relative) external setpoints of the user.

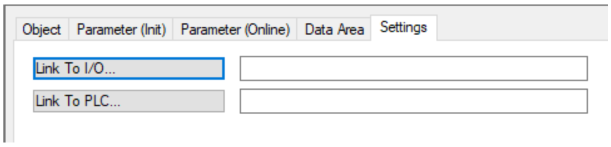
From version V3.2.60: The "CoordinateSystemOID" parameter specifies the coordinate system that SetPos and ActPos are specified in and the coordinate system that the mover is located in.

Data Area: Shows memory areas via which the mover is linked to other objects and exchanges information.



	Area No	Name	Type	Size	CS	CD / Elements	Owner
+	4 (0)	IoToMc	InputDst	8	☑	□ 1 Symbols	
+	0 (0)	PlcToMc	InputDst	8	☑	□ 1 Symbols	05100010, Offs: 0
+	5 (0)	McToIo	OutputSrc	8	☑	□ 1 Symbols	
+	1 (0)	McToPlc	OutputSrc	40	☑	□ 5 Symbols	05100010, Offs: 0

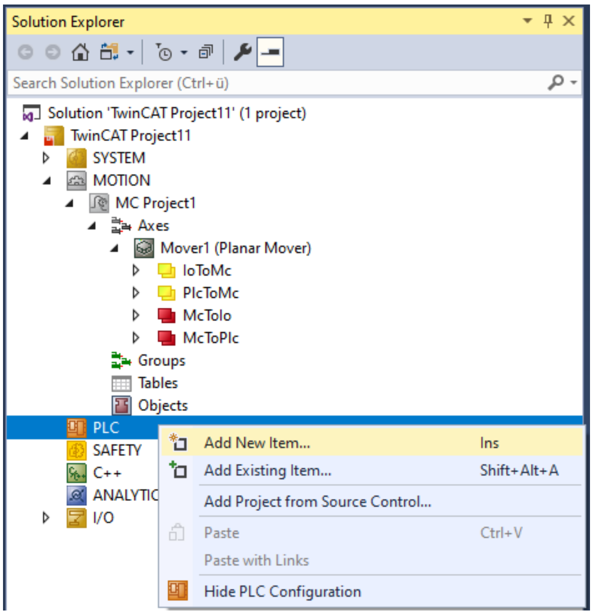
Settings: The user can establish links here. With the two "Link To ..." buttons, the Planar mover can be linked to the movers in the PLC and the XPlanar driver.



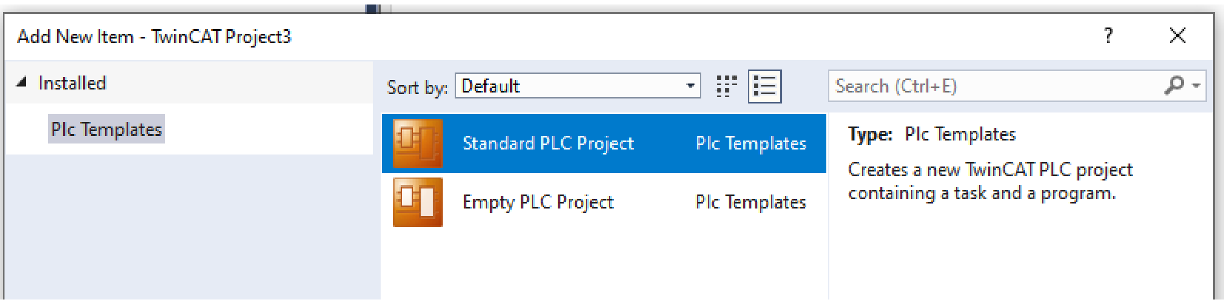
6.1.2 Creating a PLC

✓ A PLC must be created to control the mover, track or group, to create the geometry of an environment or to use Planar Feedback.

1. Select **PLC > Add New Item...**



2. In the following dialog box, select **Standard PLC Project** and confirm with **OK**.


Add New Item - TwinCAT Project3
?
Installed
Plc Templates
Sort by: Default
Search (Ctrl+E)
Standard PLC Project Plc Templates
Empty PLC Project Plc Templates
Type: Plc Templates
Creates a new TwinCAT PLC project containing a task and a program.

3. Add the libraries "Tc3_Physics" and "Tc3_Mc3PlanarMotion" to the PLC project; see Inserting libraries [▸ 125].

⇨ The PLC is created and you can issue commands to the corresponding objects as described in the following examples.

6.1.3 Example: "Creating and moving Planar movers"

Using this short guide you will create a TwinCAT project that contains a Planar mover and moves it in a simple way.

Creating a Planar mover

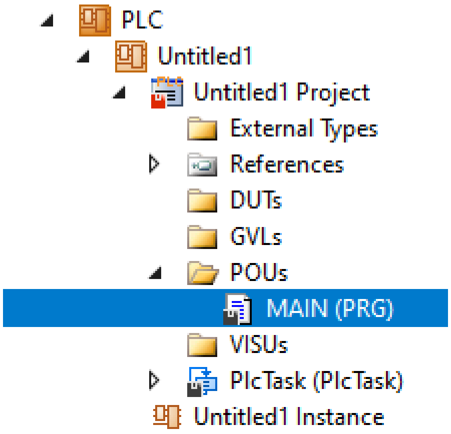
✓ See Configuration [▶ 18].

1. Create a Planar mover for this example.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a PLC

✓ See preliminary steps Creating a PLC [▶ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▶ 143]") as follows.



⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a state variable for a state machine and a target position for a travel command of the mover, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    state : UDINT;
    target_position : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the mover and moves it to position x = 100 and y = 100.

```
CASE state OF
  0:
    mover.Enable(0);
    state := 1;
  1:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    target_position.SetValuesXY(100, 100);
    mover.MoveToPosition(0, target_position, 0, 0);
    state := 3;
END_CASE
```

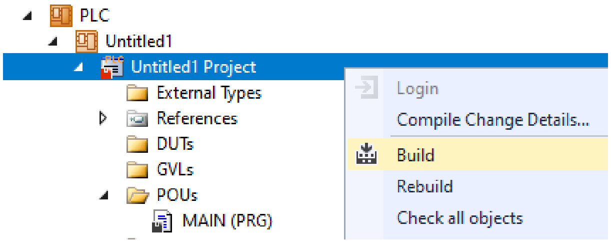
Sending the command

4. To send the command, you must call the mover cyclically with its update method after END_CASE:

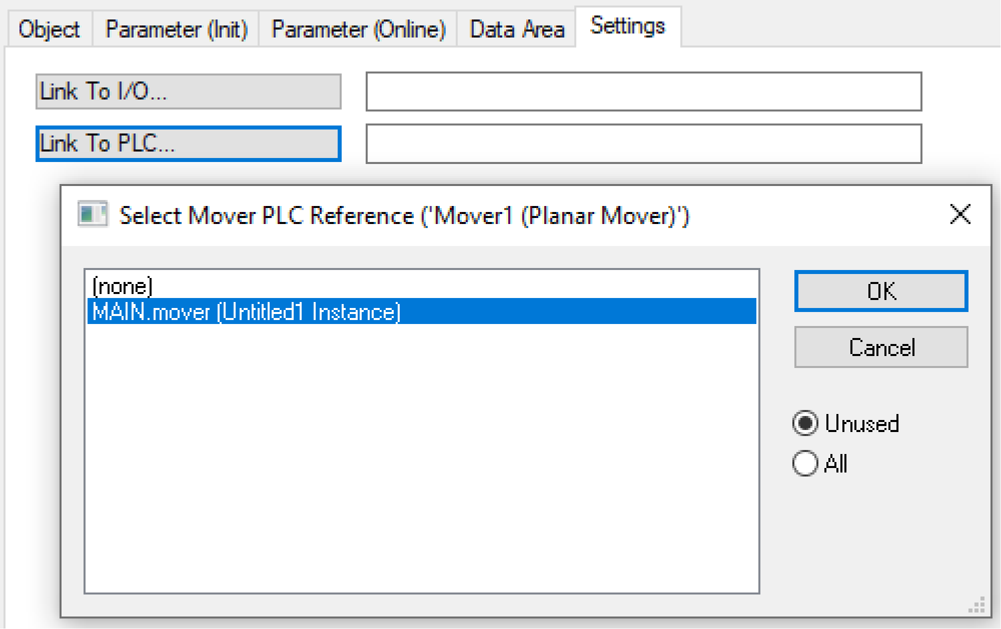
```
mover.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



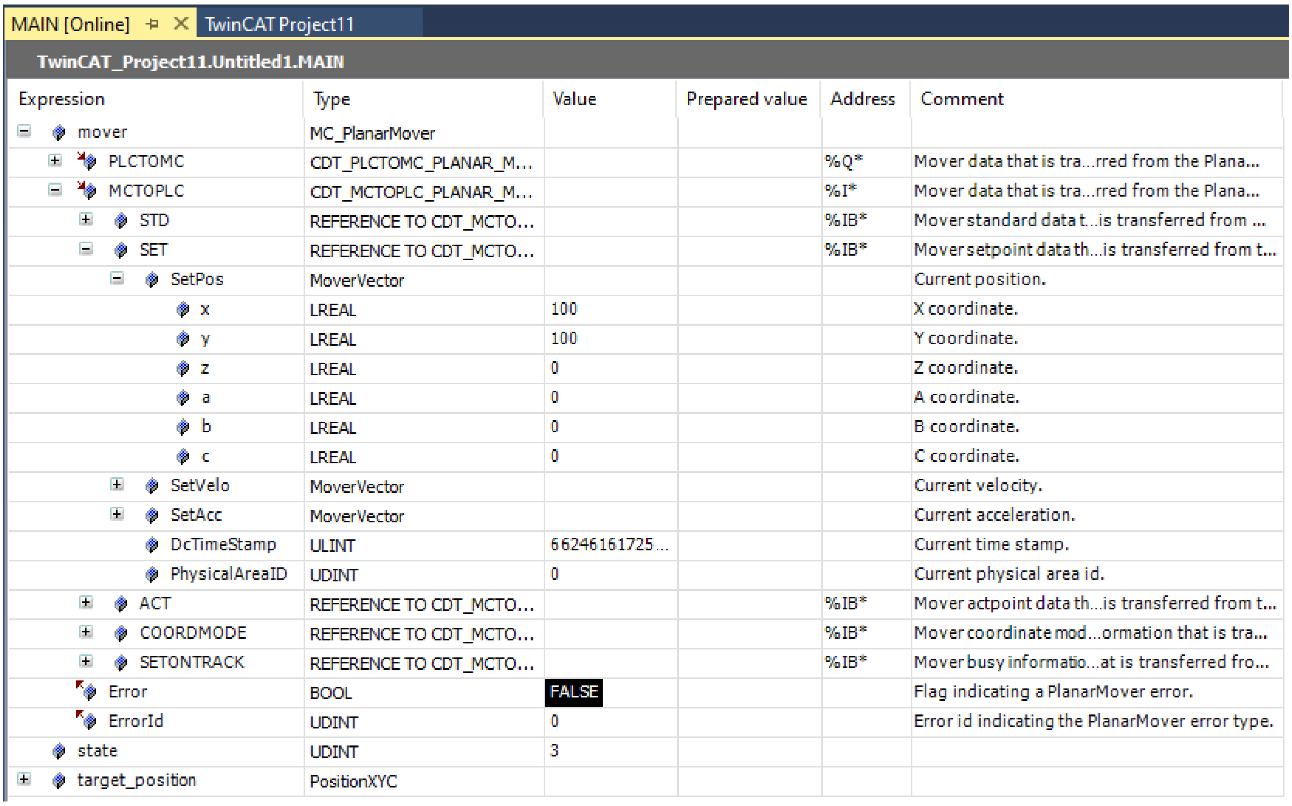
⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=3), the mover is in the desired position.



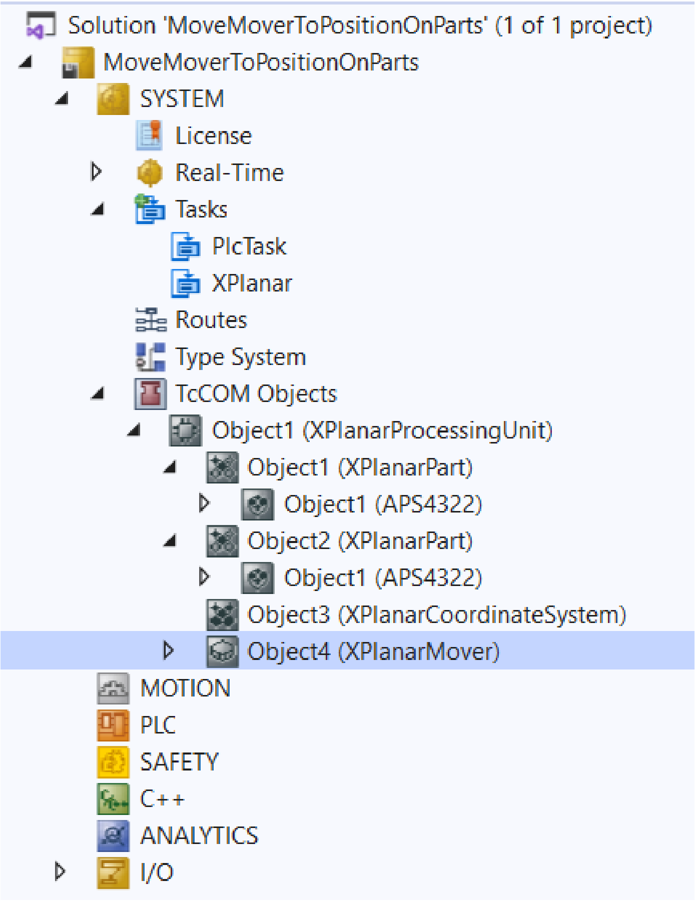
Expression	Type	Value	Prepared value	Address	Comment
mover	MC_PlanarMover				
PLCTOMC	CDT_PLCTOMC_PLANAR_M...			%Q*	Mover data that is tra...rred from the Plana...
MCTOPLC	CDT_MCTOPLC_PLANAR_M...			%I*	Mover data that is tra...rred from the Plana...
STD	REFERENCE TO CDT_MCTO...			%IB*	Mover standard data t...is transferred from ...
SET	REFERENCE TO CDT_MCTO...			%IB*	Mover setpoint data th...is transferred from t...
SetPos	MoverVector				Current position.
x	LREAL	100			X coordinate.
y	LREAL	100			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
SetVelo	MoverVector				Current velocity.
SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	66246161725...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
ACT	REFERENCE TO CDT_MCTO...			%IB*	Mover actpoint data th...is transferred from t...
COORDMODE	REFERENCE TO CDT_MCTO...			%IB*	Mover coordinate mod...ormation that is tra...
SETONTRACK	REFERENCE TO CDT_MCTO...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
state	UDINT	3			
target_position	PositionXYC				

6.1.4 Example "Moving a Planar mover to Planar parts"

In this example, a Planar mover is moved onto two Planar parts.

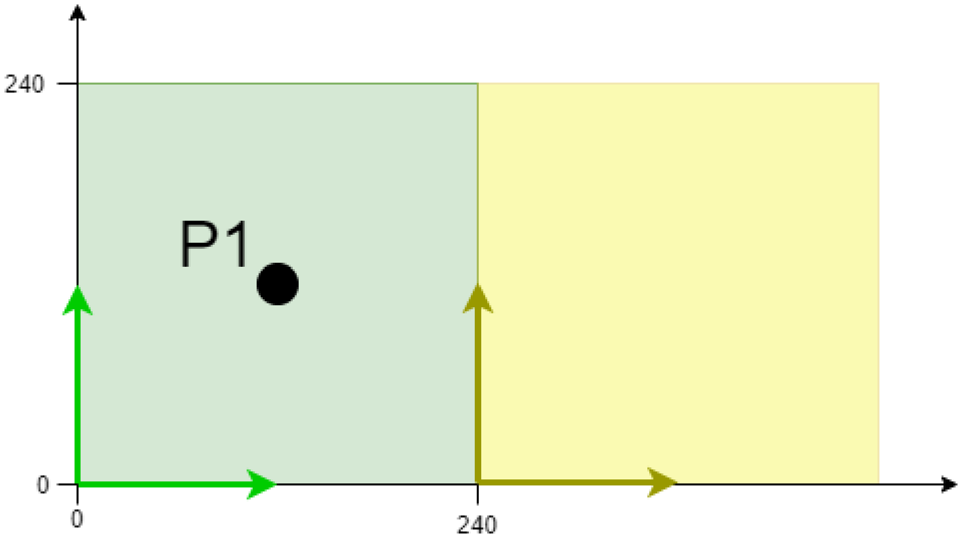
Starting point

You start with a solution that contains a fully configured XPlanar Processing Unit. Two parts, a coordinate system and a mover are created under the XPlanar Processing Unit. A tile is created under each of the two parts.



- Solution 'MoveMoverToPositionOnParts' (1 of 1 project)
  - MoveMoverToPositionOnParts
    - SYSTEM
      - License
      - Real-Time
      - Tasks
        - PlcTask
        - XPlanar
      - Routes
      - Type System
      - TcCOM Objects
        - Object1 (XPlanarProcessingUnit)
          - Object1 (XPlanarPart)
            - Object1 (APS4322)
          - Object2 (XPlanarPart)
            - Object1 (APS4322)
          - Object3 (XPlanarCoordinateSystem)
          - Object4 (XPlanarMover)
    - MOTION
    - PLC
    - SAFETY
    - C++
    - ANALYTICS
    - I/O

The following geometric situation is set: the two parts are next to each other and the mover starts in the middle of the left part (position P1). Both parts are not movable and the configuration is therefore static.



The example is developed on the basis of this configuration.

The creation of the initial situation is described in the XPlanar Processing Unit documentation.

Creating a Planar mover and a Planar environment

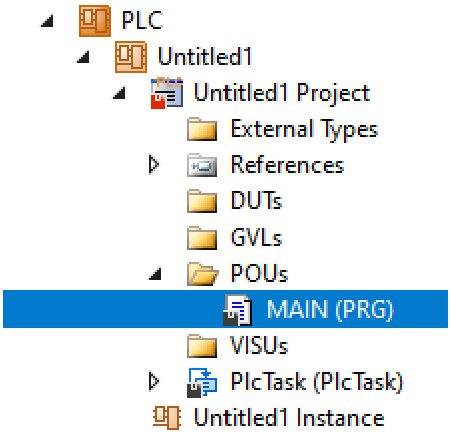
1. Create a Planar mover for this example, see Configuration [▸ 18].
2. Create a Planar environment, see Configuration [▸ 96].

3. Set the initial parameter XPlanar processing unit OID to the object ID of the XPlanar Processing Unit. This activates the Part feature for all **MC Configuration** objects (especially for the created Planar mover).

Creating a PLC

✓ See preliminary steps Creating a PLC [▶ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▶ 143]") as follows.



⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a state variable for a state machine and a target position for a travel command of the mover, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    state : UDINT;
    target_position : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the mover and moves it to the position x=100 and y=100, which is specified in the part coordinate system of the right part (its object Id is 16#01010030).

```
CASE state OF
  0:
    mover.Enable(0);
    state := 1;
  1:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    target_position.SetValuesXYCReferenceId(100, 100, 0, 16#01010030);
    mover.MoveToPosition(0, target_position, 0, 0);
    state := 3;
END_CASE
```

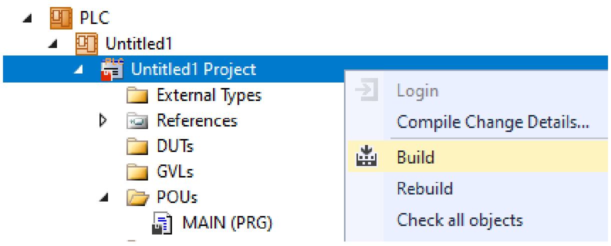
Sending the command

4. To send the command, you must call the mover cyclically with its update method after END_CASE:

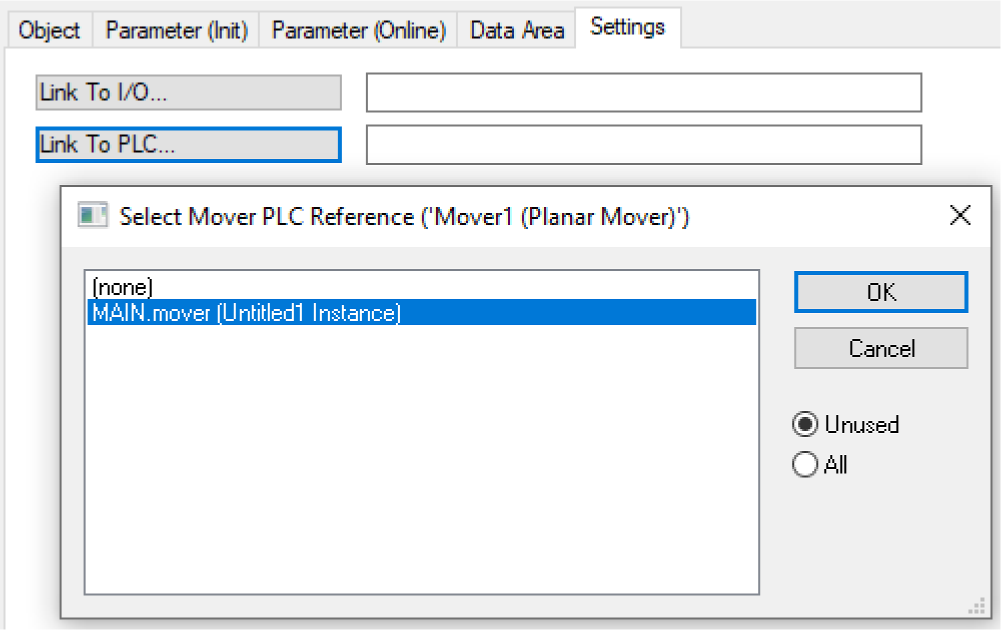
```
mover.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

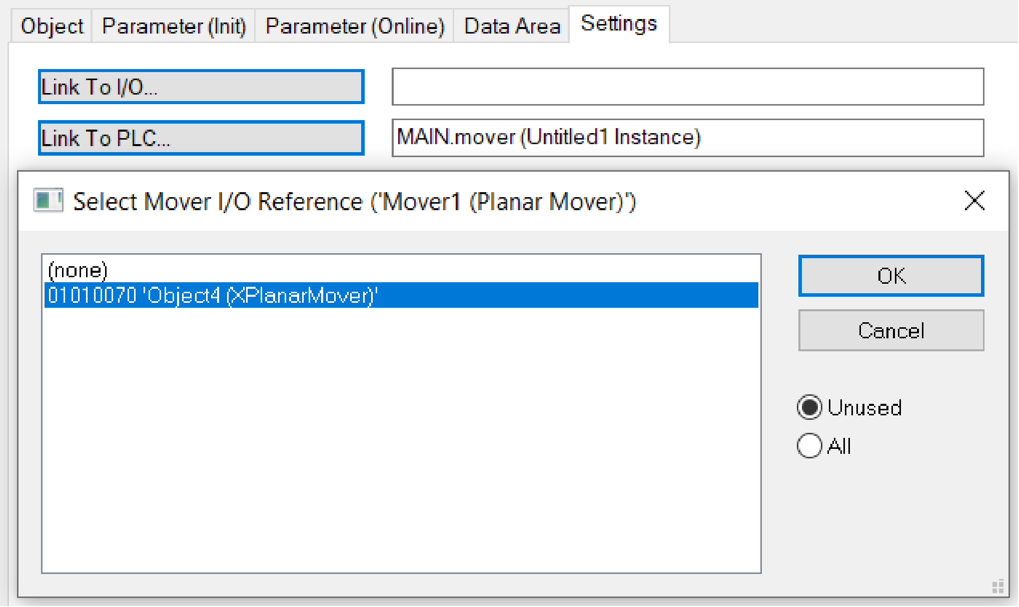
1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



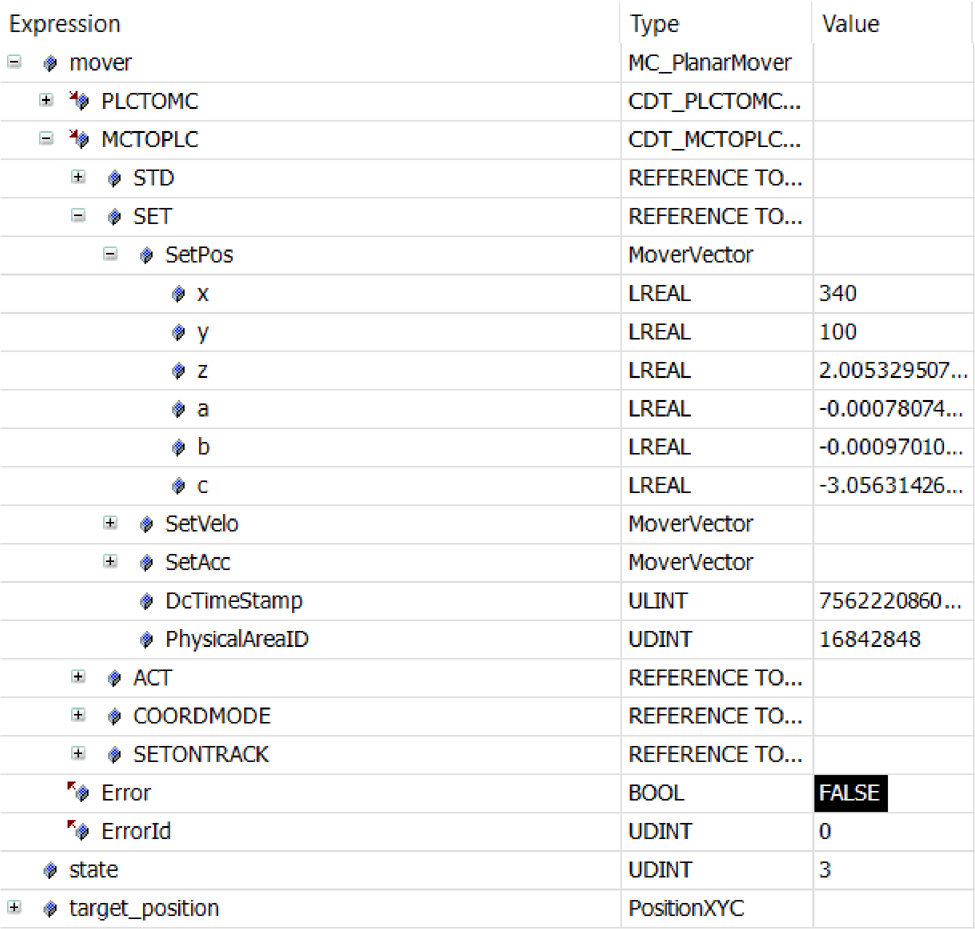
⇨ In addition, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To I/O...** button on the **Settings** tab.


Object
Parameter (Init)
Parameter (Online)
Data Area
Settings
Link To I/O...
Link To PLC...
MAIN.mover (Untitled1 Instance)
Select Mover I/O Reference ('Mover1 (Planar Mover)')
(none)
01010070 'Object4 (XPlanarMover)'
OK
Cancel
● Unused
○ All

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=3), the mover is in the desired position. The position is specified in the coordinate system and not, like the target_position, in the part coordinate system of the right-hand part (its object Id is 16#01010030). Both systems are shifted by 240 mm in the x-direction. It can also be seen that the axes a, b and z exhibit a slight noise. This is generated by the simulation of the XPlanar Processing Unit and has an effect on the initially accepted position when starting up.



Expression	Type	Value
mover	MC_PlanarMover	
PLCTOMC	CDT_PLCTOMC...	
MCTOPLC	CDT_MCTOPLC...	
STD	REFERENCE TO...	
SET	REFERENCE TO...	
SetPos	MoverVector	
x	LREAL	340
y	LREAL	100
z	LREAL	2.005329507...
a	LREAL	-0.00078074...
b	LREAL	-0.00097010...
c	LREAL	-3.05631426...
SetVelo	MoverVector	
SetAcc	MoverVector	
DcTimeStamp	ULINT	7562220860...
PhysicalAreaID	UDINT	16842848
ACT	REFERENCE TO...	
COORDMODE	REFERENCE TO...	
SETONTRACK	REFERENCE TO...	
Error	BOOL	FALSE
ErrorId	UDINT	0
state	UDINT	3
target_position	PositionXYC	

6.1.5 Example: "Creating and moving a Planar mover with auxiliary axes"

Using this short guide you will create a TwinCAT project that contains a Planar mover and moves it in a simple way.

Creating a Planar mover

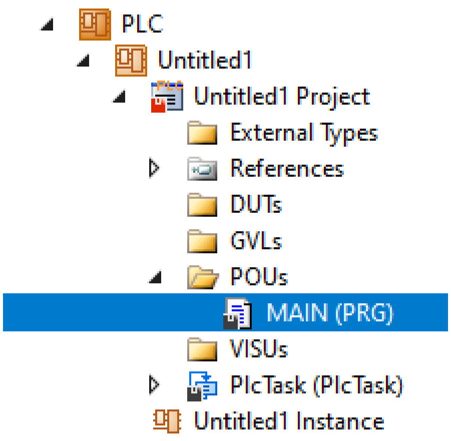
✓ See Configuration [▶ 18].

1. Create a Planar mover.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a PLC

✓ See preliminary steps Creating a PLC [▶ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▶ 143]") as follows.



⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a state variable for a state machine and a target position for a travel command of the mover, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    state : UDINT;
    target_a : LREAL := 1.0;
    target_b : LREAL := -1.0;
    target_c : LREAL := 3.0;
    target_z : LREAL := 5.0;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the mover and moves the four auxiliary axes.

```
CASE state OF
  0:
    mover.Enable(0);
    state := 1;
  1:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    mover.MoveA(0, target_a, 0);
    mover.MoveB(0, target_b, 0);
    // Since Version V3.1.10.11 MoveC has an options parameter,
    // details can be found in the CRotation example
    // and the options descriptions
    //mover.MoveC(0, target_c, 0);  // until version V3.1.10.11
    mover.MoveC(0, target_c, 0, 0); // since version V3.1.10.30
    mover.MoveZ(0, target_z, 0);
    state := 3;
END_CASE
```

Further information:

- Example "Moving the Planar mover in CRotation mode" [▶ 38]
- Limits and options of the motion commands [▶ 40]

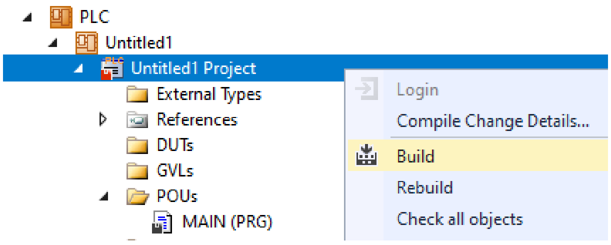
Sending the command

4. To send the command, you must call the mover cyclically with its update method after END_CASE:

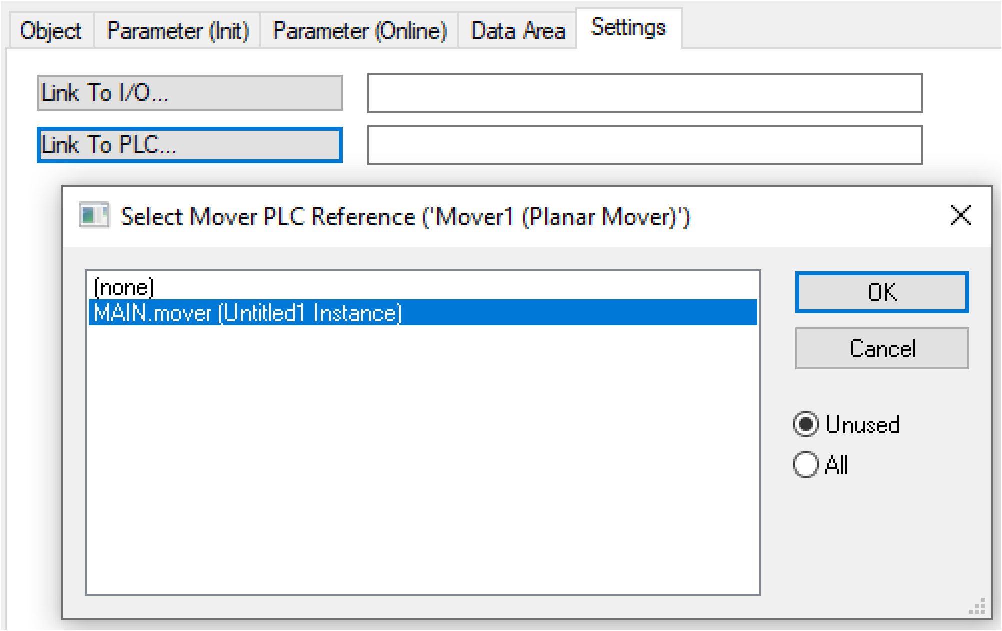
```
mover.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



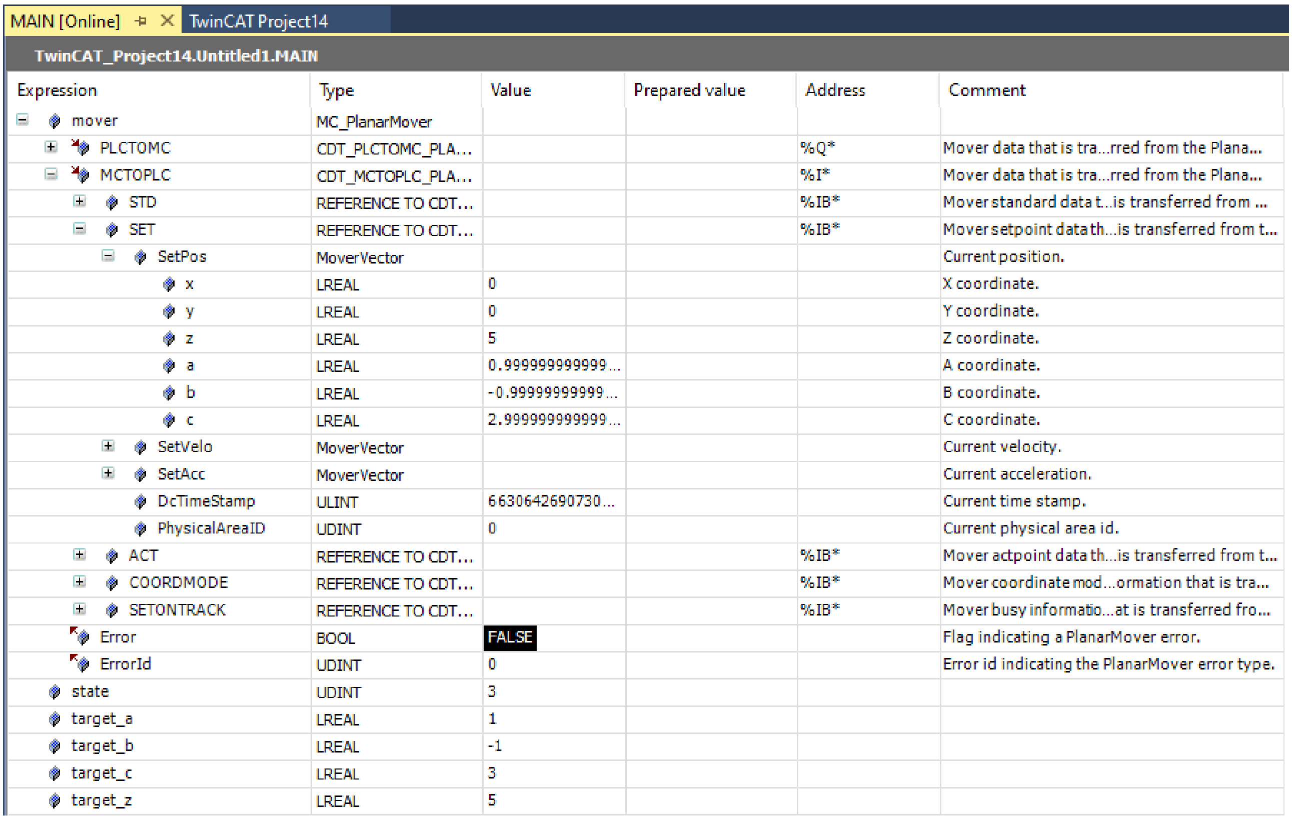
⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=3), the mover is in the desired position.



MAIN [Online] TwinCAT Project14

TwinCAT_Project14.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
⊟ mover	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	0			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	5			Z coordinate.
a	LREAL	0.999999999999...			A coordinate.
b	LREAL	-0.99999999999...			B coordinate.
c	LREAL	2.999999999999...			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6630642690730...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
state	UDINT	3			
target_a	LREAL	1			
target_b	LREAL	-1			
target_c	LREAL	3			
target_z	LREAL	5			

6.1.6 Example "Creating and moving a Planar mover with External Setpoint Generation"

Using this short guide you will create a TwinCAT project that contains a Planar mover and moves it in a simple way by means of external setpoint generation.

Creating a Planar mover

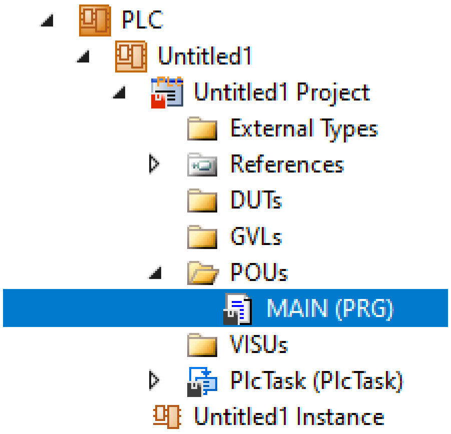
✓ See Configuration [▶ 18].

1. Create a Planar mover.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a PLC

✓ See preliminary steps Creating a PLC [▶ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▶ 143]") as follows.



⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a state variable for a state machine and variables for the external setpoint, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    state : UDINT;
    p,v,a : MoverVector;
    deltat : LREAL := 0.001;
velo, acc, jerk : LREAL;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the mover and starts the external setpoint generation. A profile is then followed that ends with a positive velocity. The subsequent stopping of the external setpoint generation ensures that the mover reduces its velocity to zero and is in the FreeMovement state after stopping (this is done with the maximum dynamics of the mover).

```
CASE state OF
  0:
    mover.Enable(0);
    state := 1;
  1:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    p.x := 0.0; v.x := 0.0; a.x := 0.0;
    mover.StartExternalSetpointGeneration(0,0);
    mover.SetExternalSetpoint(p,v,a);
    state := 3;
  3:
    velo := v.x;
    acc := a.x;
    p.x := p.x + deltat * velo + deltat * deltat / 2 * acc + deltat * deltat * deltat / 6 *
jerk;
    v.x := v.x + deltat * acc + deltat * deltat / 2 * jerk;
    a.x := a.x + deltat * jerk;
    mover.SetExternalSetpoint(p,v,a);
    IF a.x >= 10.0 THEN
      jerk := -1;
    END_IF;
    IF a.x <= 0.0 THEN
      state := 4;
    END_IF;
  5:
    mover.StopExternalSetpointGeneration(0);
    state := 6;
END_CASE
```

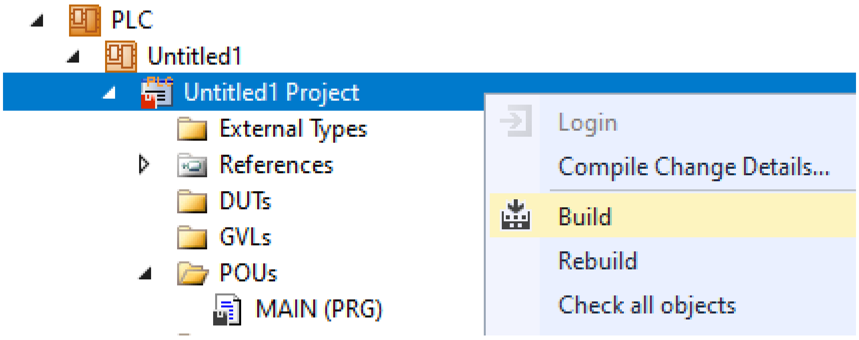
Sending the command

4. To send the commands you need to trigger the update method of the mover after the END_CASE:

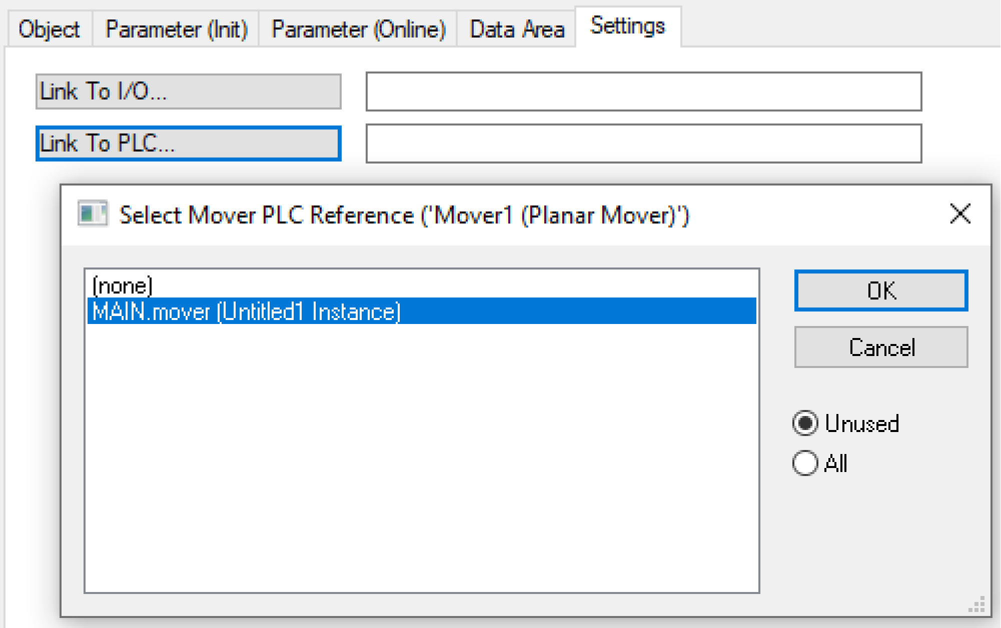
```
mover.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



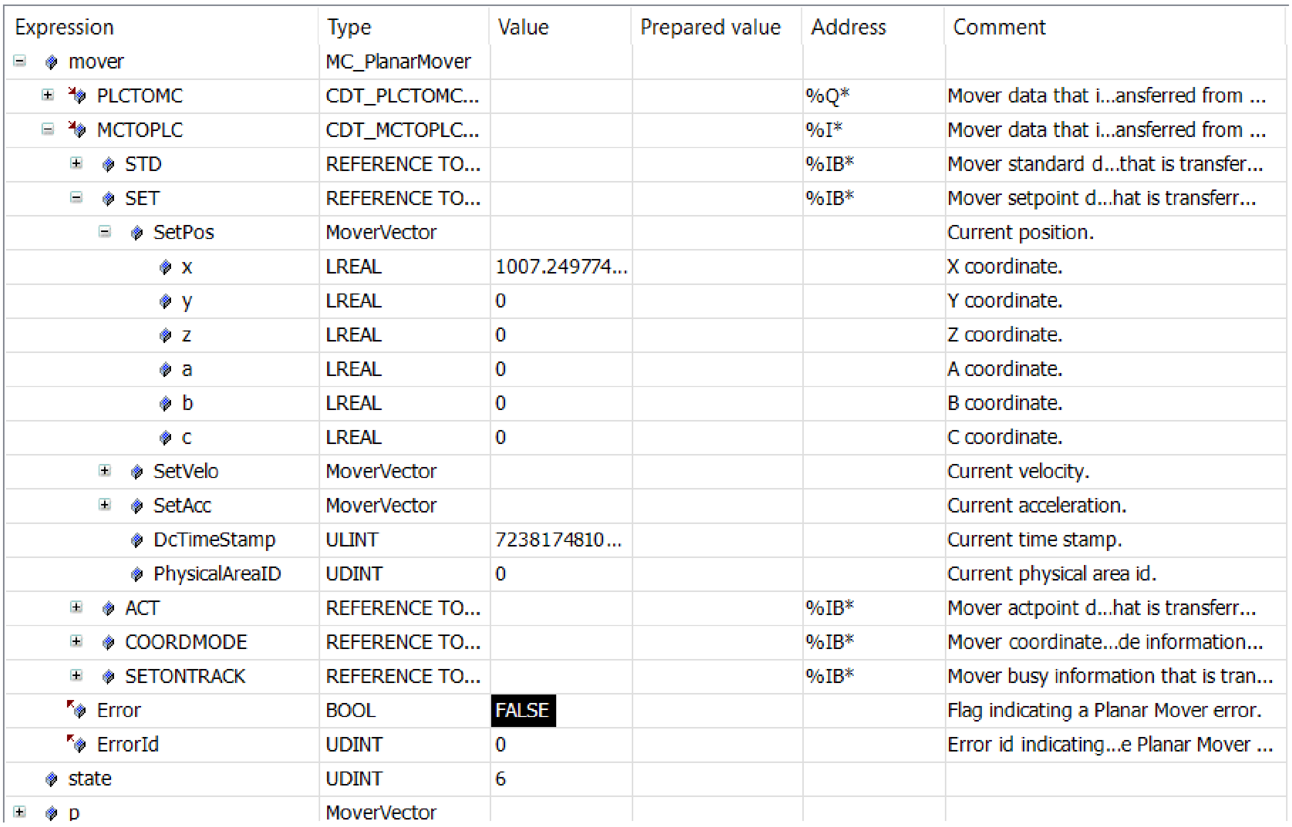
⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state = 6), the mover is in the desired positive x-position.



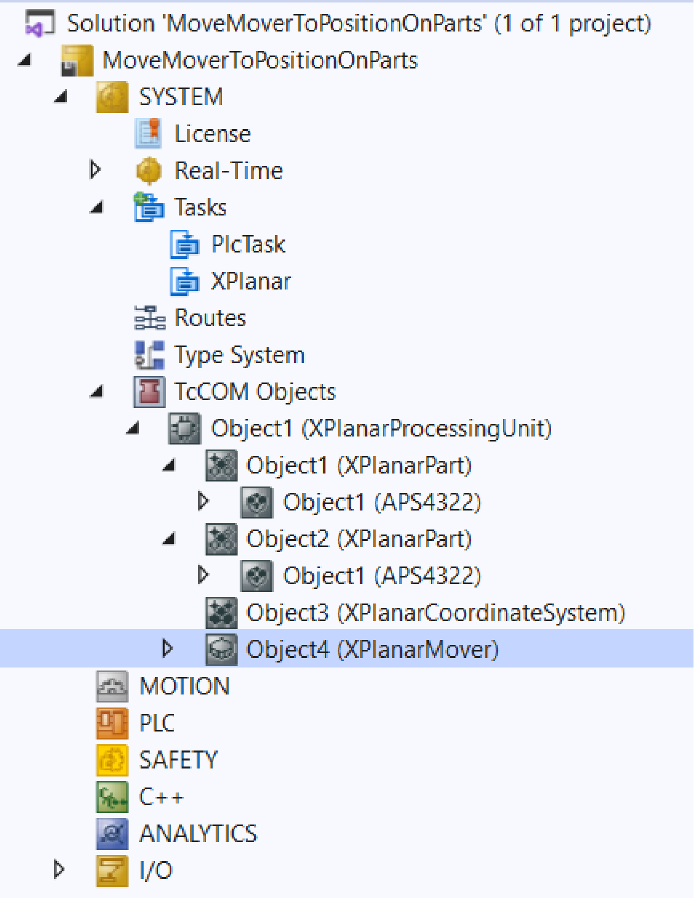
Expression	Type	Value	Prepared value	Address	Comment
⊟ mover	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC...			%Q*	Mover data that i...ansferred from ...
⊟ MCTOPLC	CDT_MCTOPLC...			%I*	Mover data that i...ansferred from ...
⊞ STD	REFERENCE TO...			%IB*	Mover standard d...that is transfer...
⊟ SET	REFERENCE TO...			%IB*	Mover setpoint d...hat is transferr...
⊟ SetPos	MoverVector				Current position.
x	LREAL	1007.249774...			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	7238174810...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO...			%IB*	Mover actpoint d...hat is transferr...
⊞ COORDMODE	REFERENCE TO...			%IB*	Mover coordinate...de information...
⊞ SETONTRACK	REFERENCE TO...			%IB*	Mover busy information that is tran...
Error	BOOL	FALSE			Flag indicating a Planar Mover error.
ErrorId	UDINT	0			Error id indicating...e Planar Mover ...
state	UDINT	6			
⊞ p	MoverVector				

6.1.7 Example "Moving a Planar mover with External Setpoint Generation to Planar parts"

In this example, a Planar mover is moved to two Planar parts with external setpoint generation.

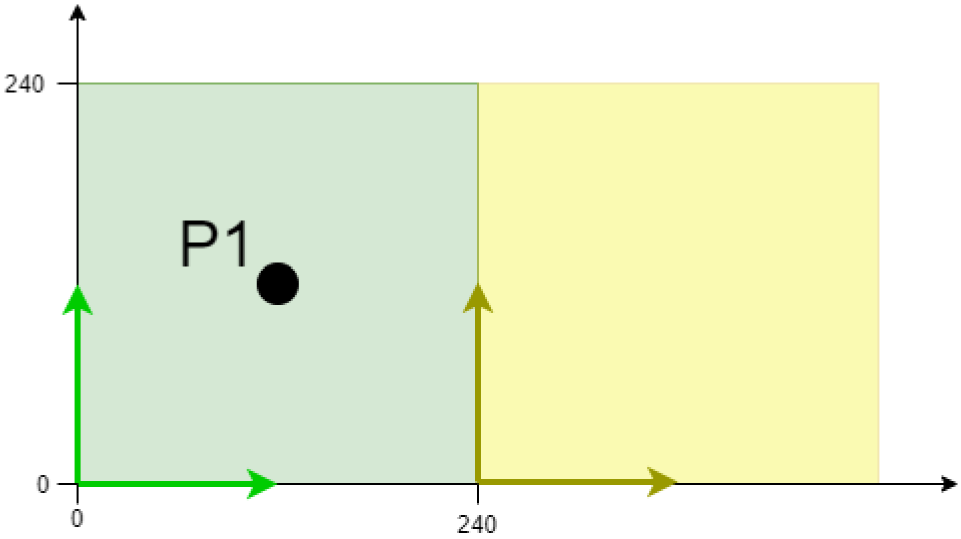
Starting point

You start with a solution that contains a fully configured XPlanar Processing Unit. Two parts, a coordinate system and a mover are created under the XPlanar Processing Unit. A tile is created under each of the two parts.



```
Solution 'MoveMoverToPositionOnParts' (1 of 1 project)
◢ MoveMoverToPositionOnParts
  ◢ SYSTEM
      License
    ▷ Real-Time
    ◢ Tasks
        PlcTask
        XPlanar
      Routes
      Type System
    ◢ TcCOM Objects
      ◢ Object1 (XPlanarProcessingUnit)
        ◢ Object1 (XPlanarPart)
          ▷ Object1 (APS4322)
        ◢ Object2 (XPlanarPart)
          ▷ Object1 (APS4322)
          Object3 (XPlanarCoordinateSystem)
        ▷ Object4 (XPlanarMover)
    MOTION
    PLC
    SAFETY
    C++
    ANALYTICS
  ▷ I/O
```

The following geometric situation is set: the two parts are next to each other and the mover starts in the middle of the left part (position P1). Both parts cannot be moved; the configuration is therefore static.



The example is developed on the basis of this configuration.

The creation of the initial situation is described in the XPlanar Processing Unit documentation.

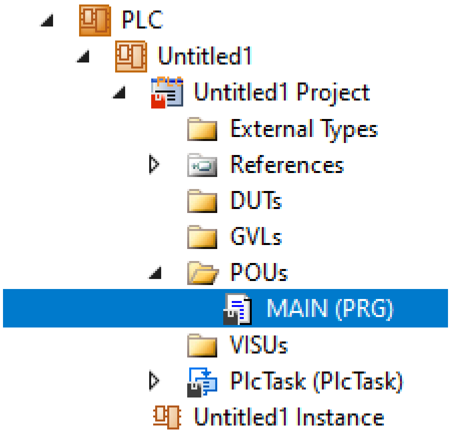
Creating a Planar mover and a Planar environment

1. Create a Planar mover for this example, see Configuration [▶ 18].
2. Create a Planar environment, see Configuration [▶ 96].
3. Set the initial parameter XPlanar processing unit OID to the object ID of the XPlanar Processing Unit. This activates the Part feature for all **MC Configuration** objects (especially for the created Planar mover).

Creating a PLC

✓ See preliminary steps Creating a PLC [▶ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▶ 143]") as follows.



PLC
Untitled1
Untitled1 Project
External Types
References
DUTs
GVLs
POUs
MAIN (PRG)
VISUs
PlcTask (PlcTask)
Untitled1 Instance

⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a state variable for a state machine and variables for the external setpoint, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    state : UDINT;
    p,v,a : MoverVector;
    deltat : LREAL := 0.01;
    refsys : OTCID := 0;
    velo, acc, jerk : LREAL;
    feedback : MC_PlanarFeedback;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the mover and starts the external setpoint generation. A profile is then followed that ends with a positive velocity. The subsequent stopping of the external setpoint generation ensures that the mover reduces its velocity to zero and is in the FreeMovement state after stopping (this is done with the maximum dynamics of the mover).

```
CASE state OF
  0:
    mover.Enable(0);
    state := 1;
  1:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    p := mover.MCTOPLC.SET.SetPos;
    v.x := 0.0; a.x := 0.0;
    mover.StartExternalSetpointGeneration(0,0);
    mover.SetExternalSetpointReferenceId(feedback,p,v,a,refsys);
    jerk := 10;
    state := 3;
  3:
    velo := v.x;
    acc := a.x;
    p.x := p.x + deltat * velo + deltat * deltat / 2 * acc + deltat * deltat * deltat / 6 * 
jerk;
    v.x := velo + deltat * acc + deltat * deltat / 2 * jerk;
    a.x := acc + deltat * jerk;
    mover.SetExternalSetpointReferenceId(feedback,p,v,a,refsys);
    IF a.x >= 100.0 THEN
      jerk := -10;
    END_IF;
    IF a.x <= 0.0 THEN
      state := 4;
    END_IF;
  4:
    mover.StopExternalSetpointGeneration(0);
    state := 5;

END_CASE
```

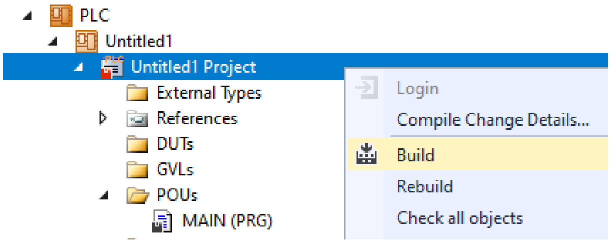
Sending the command

4. To send the commands you need to trigger the update method of the mover after the END_CASE:

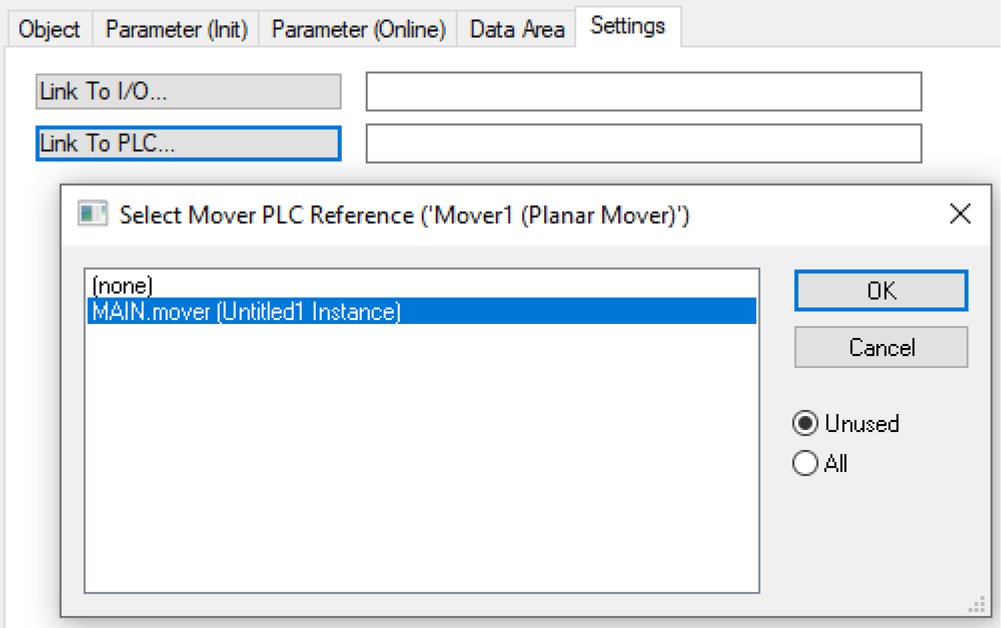
```
mover.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

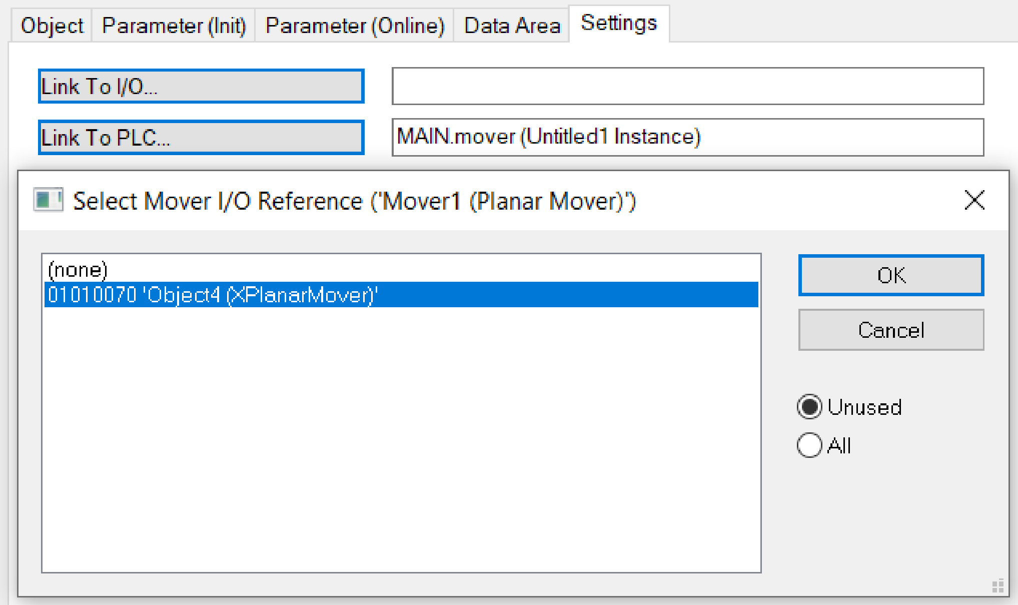
1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



⇨ In addition, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To I/O...** button on the **Settings** tab.



MAIN.mover (Untitled1 Instance)

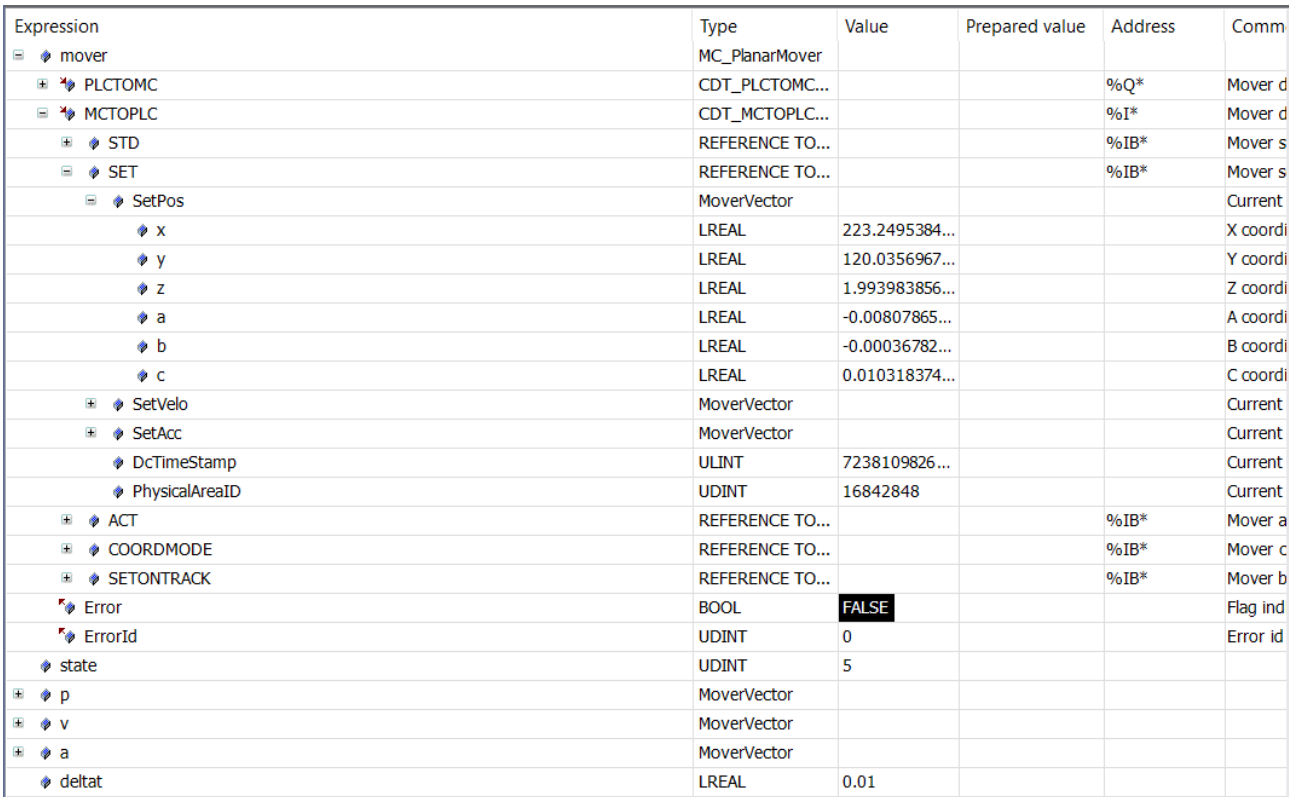
Select Mover I/O Reference ('Mover1 (Planar Mover)')

(none)
01010070 'Object4 (XPlanarMover)'

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state = 5), the mover is in the desired positive x-position. It has not left the first part, so in this case you could also specify the object ID of the first part or the coordinate system as refsys. If the object ID of the first part is specified and the limit to the second part is exceeded, the object ID of the second part must be used and the x-coordinate reduced by 240 (at the same time!). The object ID of the global coordinate system works regardless of which part you are on. As the configuration is static, zero is accepted as an alternative for the global coordinate system ID.



Expression	Type	Value	Prepared value	Address	Comm
mover	MC_PlanarMover				
PLCTOMC	CDT_PLCTOMC...			%Q*	Mover d
MCTOPLC	CDT_MCTOPLC...			%I*	Mover d
STD	REFERENCE TO...			%IB*	Mover s
SET	REFERENCE TO...			%IB*	Mover s
SetPos	MoverVector				Current
x	LREAL	223.2495384...			X coordi
y	LREAL	120.0356967...			Y coordi
z	LREAL	1.993983856...			Z coordi
a	LREAL	-0.00807865...			A coordi
b	LREAL	-0.00036782...			B coordi
c	LREAL	0.010318374...			C coordi
SetVelo	MoverVector				Current
SetAcc	MoverVector				Current
DcTimeStamp	ULINT	7238109826...			Current
PhysicalAreaID	UDINT	16842848			Current
ACT	REFERENCE TO...			%IB*	Mover a
COORDMODE	REFERENCE TO...			%IB*	Mover c
SETONTRACK	REFERENCE TO...			%IB*	Mover b
Error	BOOL	FALSE			Flag ind
ErrorId	UDINT	0			Error id
state	UDINT	5			
p	MoverVector				
v	MoverVector				
a	MoverVector				
deltat	LREAL	0.01			

Comments on the feedback:

In this example, the feedback was only created and entered in the SetExternalSetpointRefSys call without doing anything with it. As no errors occur in this example, the feedback is busy from state 2. Otherwise, you should ALWAYS implement error handling with this feedback in order to handle errors such as an incorrect RefSysId for the position.

The special feature of this feedback is that it is passed cyclically and the same feedback must be passed in every call. If errors occur in the external setpoint generation, these errors are displayed in the feedback after the next call of SetExternalSetpointRefSys with feedback. An update call for the feedback is not necessary here. In addition, the feedback is not done when StopExternalSetpointGeneration is called, or aborted when Halt is called.

Even if the information is available in the feedback after the SetExternalSetpointRefSys call, the errors do not have to come directly from this call, but can also result from previous calls.

6.1.8 Example "Moving the Planar mover in CRotationFreeMovement mode"

Using this short guide you will create a TwinCAT project that contains a Planar mover and moves it in a simple way.

Creating a Planar mover

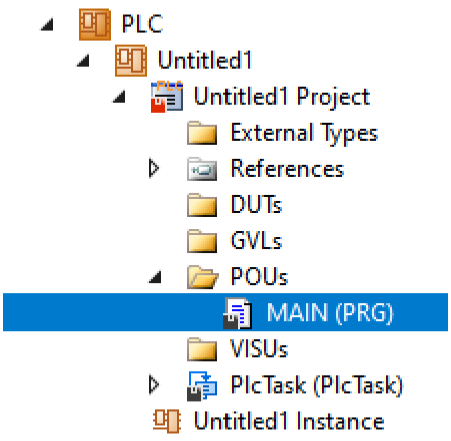
- ✓ See Configuration [▶ 18].

1. Create a Planar mover.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a PLC

- ✓ See preliminary steps Creating a PLC [▶ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▶ 143]") as follows.



- PLC
  - Untitled1
    - Untitled1 Project
      - External Types
      - References
      - DUTs
      - GVLs
      - POUs
        - MAIN (PRG)
      - VISUs
      - PlcTask (PlcTask)
    - Untitled1 Instance

⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a state variable for a state machine and a target position for a travel command of the mover, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    state : UDINT;
    target_position_c : LREAL;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the mover and rotates it to position c=20.

```
CASE state OF
  0:
    mover.Enable(0);
    state := 1;
  1:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    target_position_c := 20.0;
    mover.MoveC(0, target_position_c, 0, 0);
    state := 3;
END_CASE
```

Sending the command

4. To send the command, you must call the mover cyclically with its update method after END_CASE:

```
mover.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

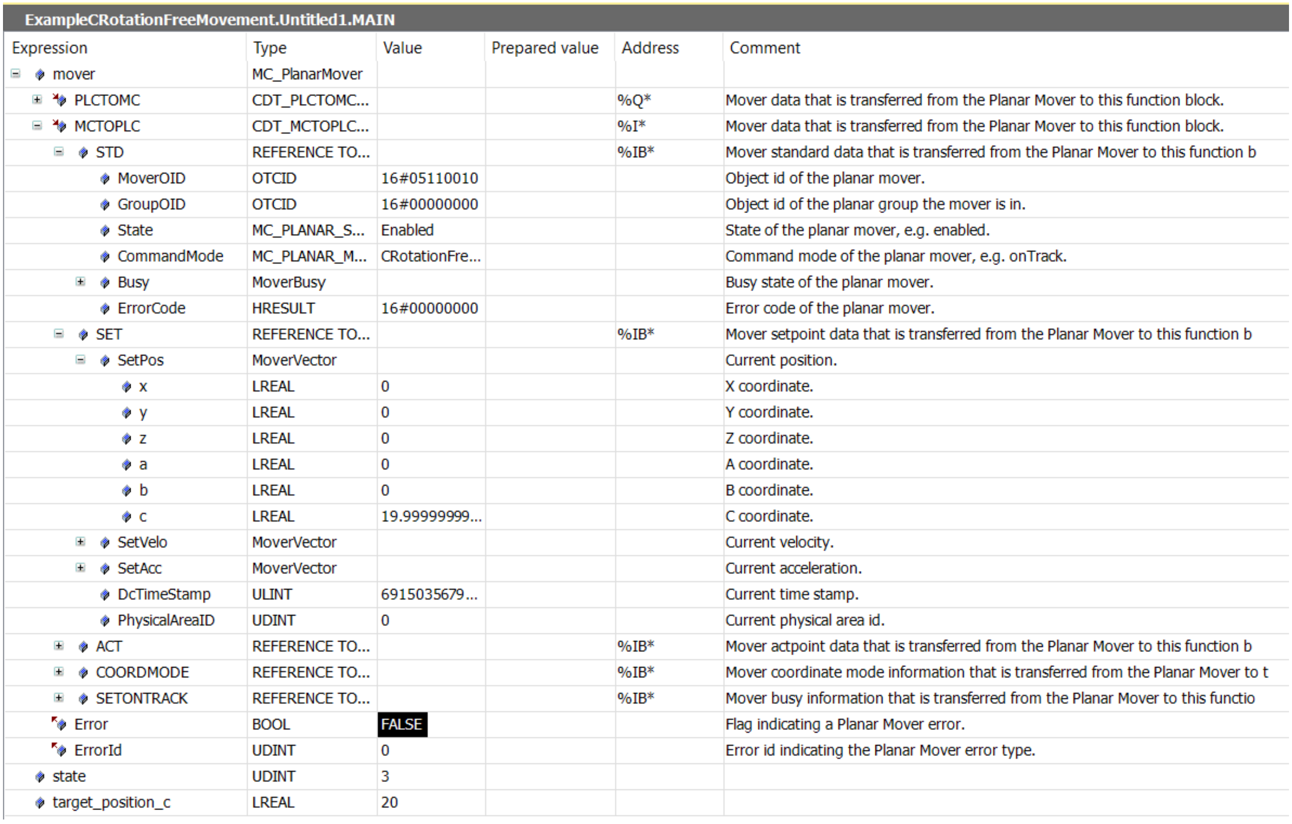
1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**

⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

The mover is at the end of the state machine (state=3) at the desired (rotated) position and is in command mode CRotationFreeMovement, since the angle is greater than 15°. A further movement of the C-axis up to e.g. 90° would change the command mode back to Free Movement after completion of the command.



ExampleCRotationFreeMovement.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
mover	MC_PlanarMover				
PLCTOMC	CDT_PLCTOMC...			%Q*	Mover data that is transferred from the Planar Mover to this function block.
MCTOPLC	CDT_MCTOPLC...			%I*	Mover data that is transferred from the Planar Mover to this function block.
STD	REFERENCE TO...			%IB*	Mover standard data that is transferred from the Planar Mover to this function b
MoverOID	OTCID	16#05110010			Object id of the planar mover.
GroupOID	OTCID	16#00000000			Object id of the planar group the mover is in.
State	MC_PLANAR_S...	Enabled			State of the planar mover, e.g. enabled.
CommandMode	MC_PLANAR_M...	CRotationFre...			Command mode of the planar mover, e.g. onTrack.
Busy	MoverBusy				Busy state of the planar mover.
ErrorCode	HRESULT	16#00000000			Error code of the planar mover.
SET	REFERENCE TO...			%IB*	Mover setpoint data that is transferred from the Planar Mover to this function b
SetPos	MoverVector				Current position.
x	LREAL	0			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	19.99999999...			C coordinate.
SetVelo	MoverVector				Current velocity.
SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6915035679...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
ACT	REFERENCE TO...			%IB*	Mover actpoint data that is transferred from the Planar Mover to this function b
COORDMODE	REFERENCE TO...			%IB*	Mover coordinate mode information that is transferred from the Planar Mover to t
SETONTRACK	REFERENCE TO...			%IB*	Mover busy information that is transferred from the Planar Mover to this functio
Error	BOOL	FALSE			Flag indicating a Planar Mover error.
ErrorId	UDINT	0			Error id indicating the Planar Mover error type.
state	UDINT	3			
target_position_c	LREAL	20			

6.1.9 Limits and options of the motion commands

The Planar mover can execute different types of motion commands. Except for the special case of external setpoint generation, these are similar in structure. The following applies to the rest of the motion commands. The first parameter of the method call is always the feedback for the command that the user transfers. If he transfers a "0", this implies that he does not want to have (or use) feedback. The next one or two parameters describe the destination of the motion and they cannot be completely omitted. The next parameters are the dynamic limits that should be observed during motion. If the user transfers a "0" here, the default values are used (TCOM parameters of the mover in the MC Project). The last parameter is the option object, which differs depending on the command.

Limits

Each motion command runs in the optimal time. For the resulting trajectory to be continuous, the time derivatives of the position must be limited. The limits include maximum values for the velocity, positive and negative acceleration, and jerk. If the values specified here exceed the maximum dynamic limits of the mover (TCOM parameters of the mover in the MC Project), they are reduced accordingly, a warning is issued and the command is executed with reduced dynamic values. There is only one Limit or Constraint object. This is understood to be a limitation of the dynamics tangential to the direction of movement of the mover.

For external setpoint generation, the only parameters are Feedback and Options.

From version V3.1.10.30: The limits should be replaced by Constraints, see Dynamics.

Options

The options vary depending on the command:

MoveToPosition/JoinTrack/LeaveTrack: The only option of these commands is the "UseOrientation" flag. This flag indicates whether or not the C coordinate of the XYC target position should also be used. If not, the C-coordinate can be moved separately via "MoveC".

MoveOnTrack: The first option is the "gap". This numerical value indicates the distance to the mover in front during the motion (and after that until the next motion command on the track). This distance is measured along the track (difference between the track positions of the two movers). Therefore, curvatures in the track must be taken into account, as they reduce the real 2D distance. The gap is calculated from center to center, therefore the width of the movers must be taken into account. The second option is "Direction", the direction of travel on the track towards the destination. This can assume the values "NonModulo" (= absolute), "Positive" (= forward), "ShortestWay" (= shortest way) and "Negative"(= backward). If the destination is reachable in the appropriate direction, the command is executed.

From version V3.1.10.30: The third option is "AdditionalTurns": the number of additional laps driven on a "Closed Loop" track with "Direction" "Positive" or "Negative". For other "Direction" cases, "AdditionalTurns" must be zero. The fourth option is "ModuloTolerance". This parameter is used to avoid unintended rotations when the start and target positions are very similar. If the distance between the start and target position is less than or equal to the "ModuloTolerance", the target position is approached by the shortest route (as with "Direction" = "Shortest Way"), i.e. against the specified "Direction". For the "Direction" "NonModulo" the ModuloTolerance must be zero. For details, see Modulo positioning.

From version V3.1.10.30: MoveC: The first option is "AdditionalTurns": the number of additional whole C-turns related to the "C coordinate modulus" parameter of the mover with "Direction" "Positive" or "Negative". For other "Direction" cases, "AdditionalTurns" must be zero. The second option is "Direction": the direction of rotation of the C coordinate towards the target. This can assume the values "NonModulo" (= absolute), "Positive" (= forward), "ShortestWay" (= shortest way) and "Negative"(= backward). For details, see Modulo positioning.

From version V3.1.10.51: AdoptTrackOrientation: The first option is "AdditionalTurns": The number of additional whole C-turns related to the "C coordinate modulus" parameter of the mover with "Direction", "Positive", or "Negative". For other "Direction" cases, "AdditionalTurns" must be zero. The second option is "Direction": the direction of rotation of the C-coordinate towards the target. This can assume the values "NonModulo" (= absolute), "Positive" (= forward), "ShortestWay" (= shortest way) and "Negative"(= backward). For details, see Modulo positioning.

From version V3.1.10.44: GearInPosOnTrack: The first option is the "Gap", which has the same interpretation here as with MoveOnTrack. The second parameter is the "InSyncToleranceDistance". It specifies how far the master and slave may move away from each other before the active Planar mover loses its synchronicity. The following two options are "Direction" and "ModuloTolerance", which both refer to the parameter "SlaveSyncPosition" (as input at the function call). These options are only available if the Planar track that the Planar mover performs its synchronization movement on is a closed loop. In this case, the interpretation of these options is analogous to that for MoveC, where here the modulus is given by the length of the Planar track. For details, see Modulo positioning. The last parameter "AllowedSlaveSyncDirections" specifies in which direction, i.e. positive (default), negative or both, the Planar mover is allowed to move during the synchronization phase. This parameter can be used, for example, to prevent a back oscillation, which would occur with the option "Both", in order to achieve the fastest possible synchronization. If the Planar mover is in sync, or has been in sync, and is currently trying to get back in sync, this parameter has no further effect.

From version V3.1.10.30: GearInPosOnTrackWithMasterMover: The first four options, "Gap", "InSyncToleranceDistance", as well as the two modulo options for the SlaveSyncPosition, are identical to the first four options of the GearInPosOnTrack command in terms of their meaning. This is followed by two parameters "Direction" and "ModuloTolerance" for the MasterSyncPosition, which are available in the same way as the modulo parameters for the SlaveSyncPosition when the Master Planar Mover is on a closed-loop track. The following option "AllowedSlaveSyncDirections" has exactly the same function as the GearInPosOnTrack command. If the last option, "FollowMover", is set, it ensures that the Slave Planar Mover does not necessarily have to obtain a Planar TrackTrail [▶ 120] to know which Planar tracks it will perform its movement over. The Slave Planar Mover will simply follow the Master Planar Mover as it moves through the network. If the Master Planar Mover and the Slave Planar Mover are on different Planar tracks when the motion command is received and the "FollowMover" option is set, the slave will try to reach the Planar track that the MasterSyncPosition is commanded on by the shortest route. In addition to the "FollowMover" option, a PlanarTrackTrail can be specified for the Slave Planar Mover. In this case, it is used to command the path to the Planar track that the MasterSyncPosition lies on (e.g. if it is to deviate from the shortest path). If it does not fully reach this Planar track, the remaining path is filled with the shortest route. If a PlanarTrackTrail is specified when the "FollowMover" option is set, the SlaveSyncPosition can be specified on a different Planar track than its initial one. In general, the following rule applies: if the "FollowMover" option is set, the Slave Planar Mover follows the master from the Planar track that the MasterSyncPosition is located on, regardless of whether a PlanarTrackTrail object has been specified.

Addition version V3.1.10.30 - Option was already available before with a different type: StartExternalSetpointGeneration: Here the user has the choice between the mode "Absolute" and "Relative". In absolute mode, the mover follows only the external setpoints specified by the user and is in ExternalSetpointGeneration command mode; otherwise, the mover is in any other command mode and adds the user's external setpoints as an offset to its current setpoint.

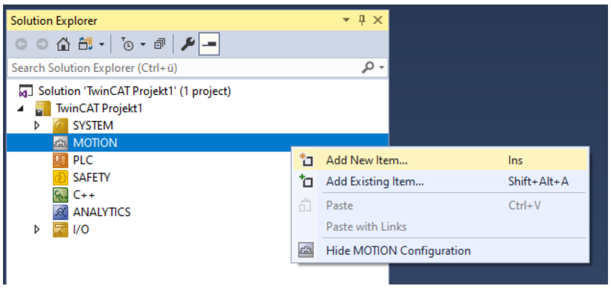
6.2 Planar track

The Planar track is a software object that represents a (virtual) one-dimensional path on the two-dimensional XPlanar stator surface. Several Planar movers can be lined up and moved on this path. Collisions are prevented by keeping a preset distance between the movers.

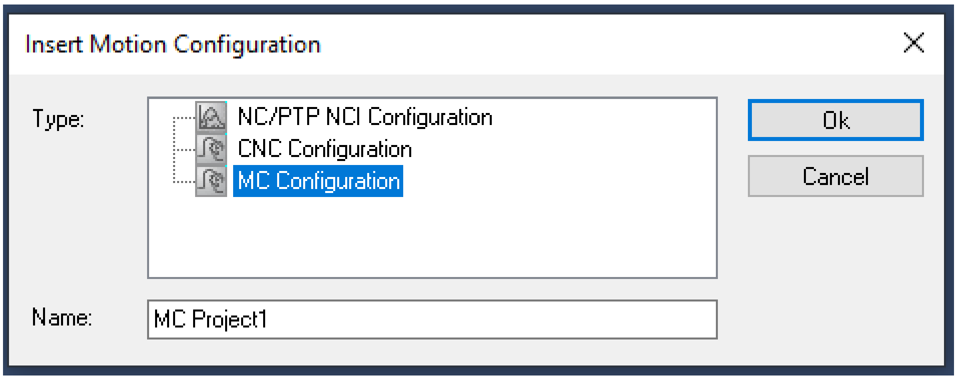
6.2.1 Configuration

✓ In order to create a Planar track, an **MC Configuration** must first be created.

1. Select **MOTION** > **Add New Item…**.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Projekt1' (1 project)
TwinCAT Projekt1
SYSTEM
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
Add New Item... Ins
Add Existing Item... Shift+Alt+A
Paste Ctrl+V
Paste with Links
Hide MOTION Configuration

2. In the following dialog box, select **MC Configuration** and confirm with **OK**.

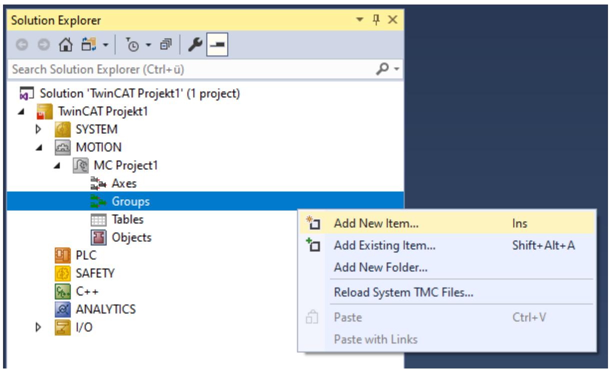


Type: MC Configuration

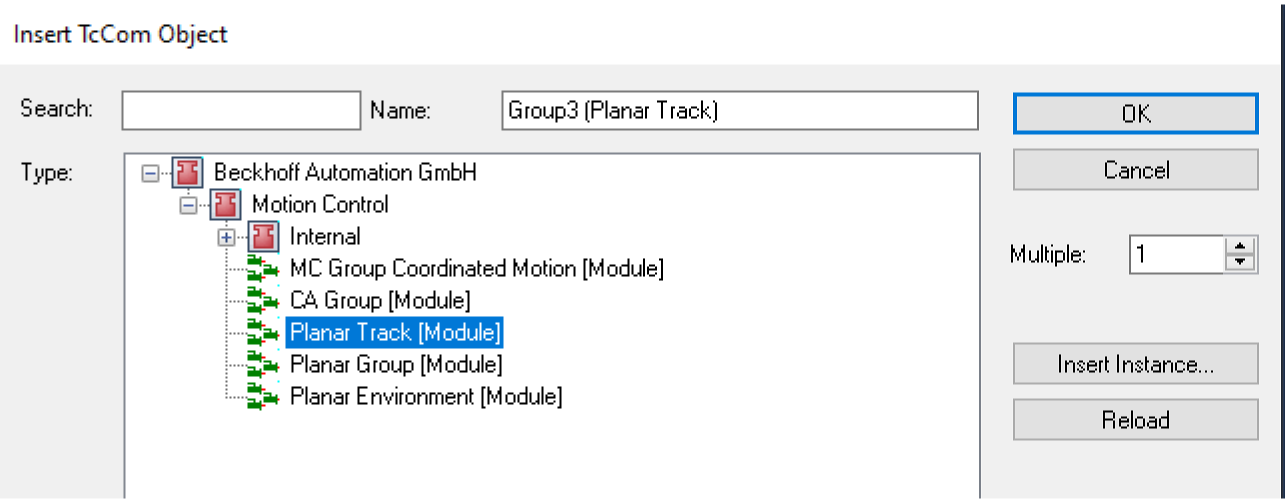
Name: MC Project1

⇨ You have created an MC Project.

3. Select **MC Project** > **Groups** > **Add New Item…**.



4. In the following dialog box, create one (or more) Planar tracks and confirm with **OK**.



Insert TcCom Object

Search:

Name: Group3 (Planar Track)

Type:

- Beckhoff Automation GmbH
  - Motion Control
    - Internal
    - MC Group Coordinated Motion [Module]
    - CA Group [Module]
    - Planar Track [Module]
    - Planar Group [Module]
    - Planar Environment [Module]

Multiple: 1

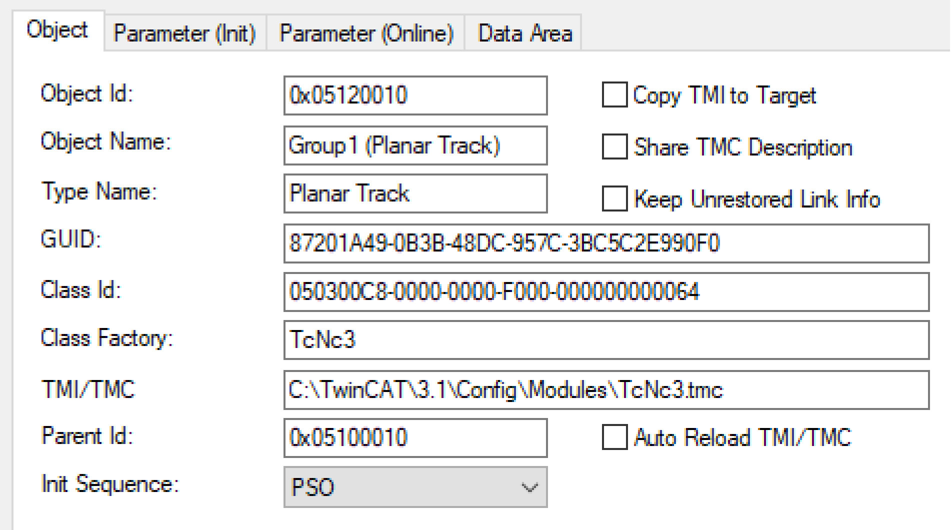
⇨ The Planar track is now created and can be parameterized.

Open detailed description

- Select the Planar track in the tree and double-click it.

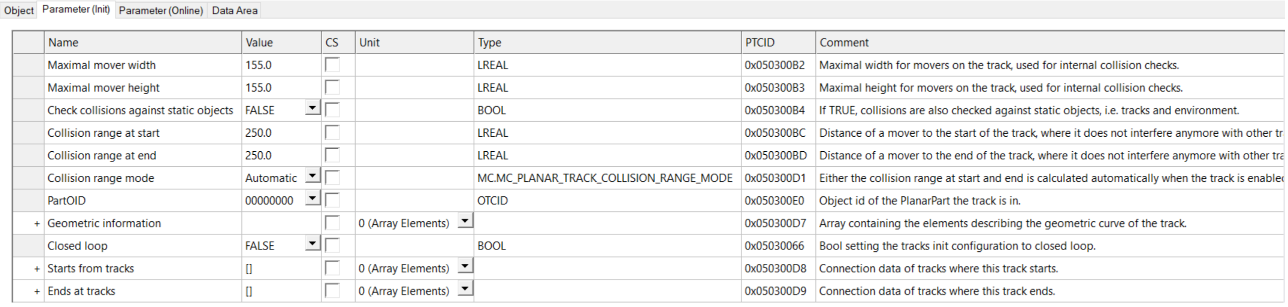
Purposes of the individual tabs

Object: General information (name, type, ID and so on) is shown here.



Object Id:	0x05120010	□ Copy TMI to Target
Object Name:	Group1 (Planar Track)	□ Share TMC Description
Type Name:	Planar Track	□ Keep Unrestored Link Info
GUID:	87201A49-0B3B-48DC-957C-3BC5C2E990F0	
Class Id:	050300C8-0000-0000-F000-000000000064	
Class Factory:	TcNc3	
TMI/TMC	C:\TwinCAT\3.1\Config\Modules\TcNc3.tmc	
Parent Id:	0x05100010	□ Auto Reload TMI/TMC
Init Sequence:	PSO	

Parameter (Init): Specifies initial parameters that the user can change in order to affect the behavior of the track.



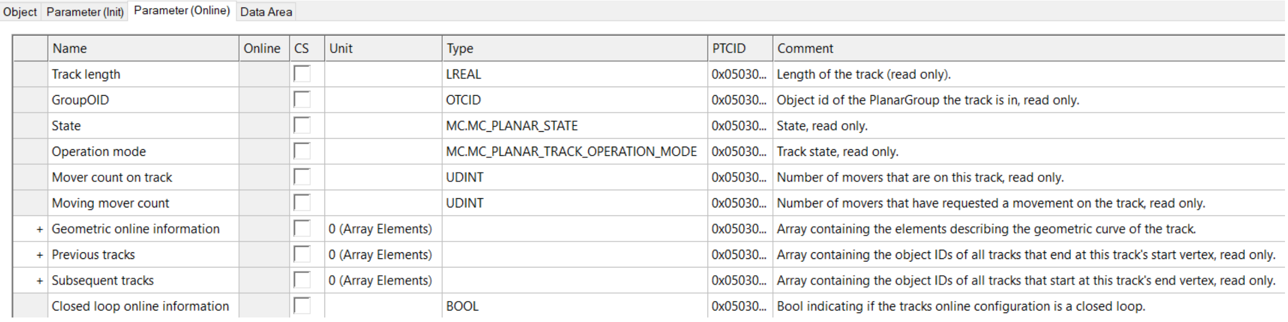
	Name	Value	CS	Unit	Type	PTCID	Comment
	Maximal mover width	155.0	□		LREAL	0x050300B2	Maximal width for movers on the track, used for internal collision checks.
	Maximal mover height	155.0	□		LREAL	0x050300B3	Maximal height for movers on the track, used for internal collision checks.
	Check collisions against static objects	FALSE	□		BOOL	0x050300B4	If TRUE, collisions are also checked against static objects, i.e. tracks and environment.
	Collision range at start	250.0	□		LREAL	0x050300BC	Distance of a mover to the start of the track, where it does not interfere anymore with other tr
	Collision range at end	250.0	□		LREAL	0x050300BD	Distance of a mover to the end of the track, where it does not interfere anymore with other tra
	Collision range mode	Automatic	□		MC.MC_PLANAR_TRACK_COLLISION_RANGE_MODE	0x050300D1	Either the collision range at start and end is calculated automatically when the track is enable
	PartOID	00000000	□		OTCID	0x050300E0	Object id of the PlanarPart the track is in.
+	Geometric information		□	0 (Array Elements)		0x050300D7	Array containing the elements describing the geometric curve of the track.
	Closed loop	FALSE	□		BOOL	0x05030066	Bool setting the tracks init configuration to closed loop.
+	Starts from tracks	[]	□	0 (Array Elements)		0x050300D8	Connection data of tracks where this track starts.
+	Ends at tracks	[]	□	0 (Array Elements)		0x050300D9	Connection data of tracks where this track ends.

The initial parameters are first set so that the Planar track (ready linked) can be traversed with the hardware. If the movers on the track are larger or smaller, the two "Maximum mover width/height" parameters should be adjusted. The parameter "Check collision against static objects" determines whether a track in a Planar group is checked for collisions with other static objects (tracks/edge of the stator surface). The parameter "Collision range mode" determines whether the "Collision range at start/end" is specified by the user via the corresponding parameters or whether it is automatically calculated internally by the track. The "Collision range" is the distance from the start/end of the track from which a Planar mover is taken into account for collision avoidance for Planar movers on other tracks.

From version V3.1.10.44: The parameters "Geometric information", "Closed loop", "Starts from tracks" and "Ends at tracks" can be used to define the geometry of the track and its connection to other tracks. The parameters act exactly like the corresponding PLC commands.

From version V3.2.60: The "PartOID" parameter specifies which part this track is permanently assigned to. If there is a unique part or the part feature is not used (no reading of the processing unit by the environment), the parameter does not need to be set. Otherwise, the parameter must be set so that the track starts. All positions in the Init and online parameters (Geometric [online] information) are specified in this part system. The "Starts from tracks" and "Ends at tracks" parameters have been extended to reflect the additional functionality of the corresponding PLC commands for the part feature.

Parameters (Online): Shows the state of the track at runtime, e.g. the number of Planar movers or the length.



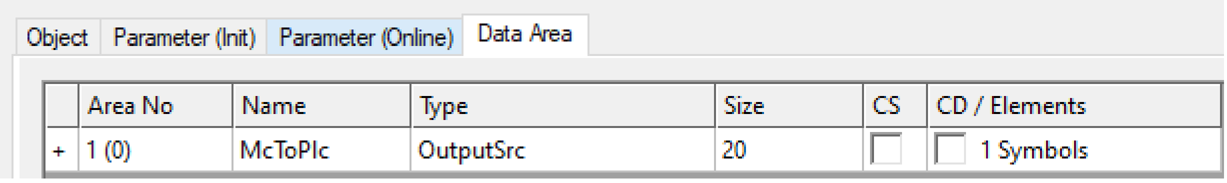
	Name	Online	CS	Unit	Type	PTCID	Comment
	Track length		□		LREAL	0x05030...	Length of the track (read only).
	GroupOID		□		OTCID	0x05030...	Object id of the PlanarGroup the track is in, read only.
	State		□		MC.MC_PLANAR_STATE	0x05030...	State, read only.
	Operation mode		□		MC.MC_PLANAR_TRACK_OPERATION_MODE	0x05030...	Track state, read only.
	Mover count on track		□		UDINT	0x05030...	Number of movers that are on this track, read only.
	Moving mover count		□		UDINT	0x05030...	Number of movers that have requested a movement on the track, read only.
+	Geometric online information		□	0 (Array Elements)		0x05030...	Array containing the elements describing the geometric curve of the track.
+	Previous tracks		□	0 (Array Elements)		0x05030...	Array containing the object IDs of all tracks that end at this track's start vertex, read only.
+	Subsequent tracks		□	0 (Array Elements)		0x05030...	Array containing the object IDs of all tracks that start at this track's end vertex, read only.
	Closed loop online information		□		BOOL	0x05030...	Bool indicating if the tracks online configuration is a closed loop.

From version V3.1.10.30: The parameters "Previous Tracks" and "Subsequent Tracks" are arrays that contain the OIDs of all tracks directly before or directly after this track.

From version V3.1.10.44: The "Geometric online information" parameter shows the geometry of the track available at runtime. This results from the corresponding initial parameter and/or the PLC commands used.

From version V3.2.1: The "Closed loop online information" parameter specifies whether the track forms a closed loop (a circle).

Data Area: Shows the memory area via which the track communicates with the PLC track.



	Area No	Name	Type	Size	CS	CD / Elements
+	1 (0)	McToPlc	OutputSrc	20	□	□ 1 Symbols

6.2.2 Track networks and collision avoidance

Tracks and track networks

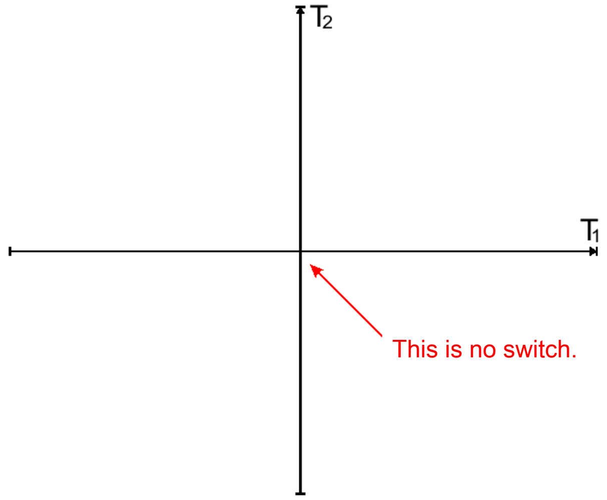
Tracks are user-specified static paths on the stator surface. Multiple tracks can be connected continuously (including direction and curvature) at one point so that movers can switch from one track to another. If more than two tracks are connected at one point in such a way, a switch is created there. This allows you to create a network of contiguous tracks.

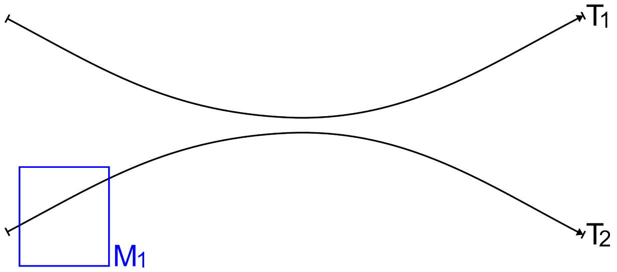
A mover can move both forward and backward on a single track. A transition to another track can only be done from a track end to a track start, not the other way around.

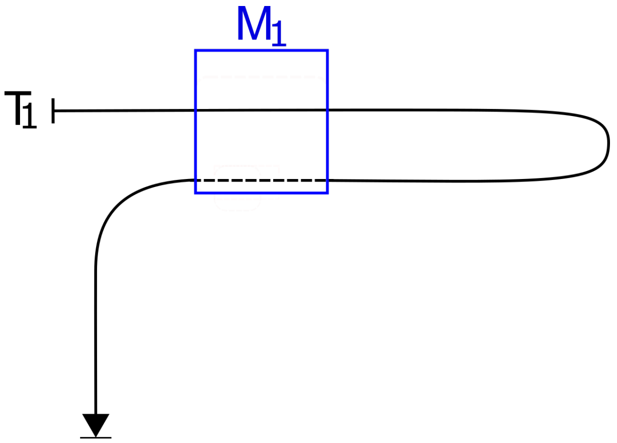
Collision avoidance in a track network

Movers that move on a track network avoid collisions with other movers in the same track network. Excluded from this are places where tracks cross without a switch or pass too close to each other or lead past themselves (see illustrations). Such configurations should be avoided.

Negative examples:







Each mover has a minimum gap set for it, which it must maintain to the mover in front of it on its path. This gap is measured between the positions of the movers on the track and can be reset with each travel command.

In the vicinity of a switch, a mover must, if necessary, additionally pay attention to potential collisions with movers that are located on other tracks connected to the switch, even if these tracks are not part of the planned path of the mover. Whether this additional collision avoidance is active for a mover at a point in time depends on four factors:

- the current position of the mover,
- the earliest possible resting position of the mover (resulting from the current dynamics and dynamic limits),
- the set gap of the mover,
- the corresponding Collision Range parameter of the current track.

If the distance between the current position and the earliest possible resting position of the mover is at any point less than Gap + Collision Range from the switch, the additional collision avoidance for this mover is active. If this is the case, all other movers for which this condition is also met are included in the dynamic planning.

Definition of the Collision Ranges

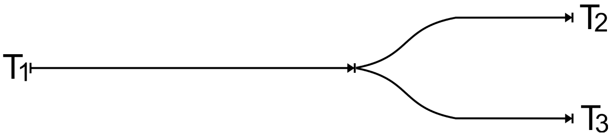
The importance of the Collision Range parameters for collision avoidance was described in the previous section. "Collision Range at start" refers to the distance to the switch at the starting point of the track and "Collision Range at end" refers to the distance to the switch at the end point of the track.

A more intuitive understanding of the Collision Range parameters arises from the following recommendation: the Collision Range should be set so that a mover that is at this distance from the associated switch (at the start or end of the track) cannot collide with movers on other tracks that connect to the switch.

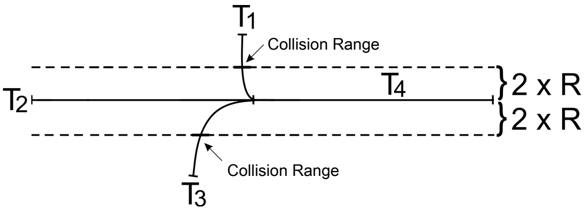
In order to simplify the configuration, the corresponding values for the Collision Ranges are automatically calculated and applied when the "Collision range mode" parameter is set to "Automatic". If "Manual" is selected instead of "Automatic", the values entered by the user are used instead. If these are set too small, this may result in collisions. If, on the other hand, they are set much too large, movers may block one another on different tracks that are actually far apart and cannot collide at all.

If a track at the starting point (end point) either has no switch, or if no other tracks start (end) at the switch, the corresponding Collision Range can be set to 0.

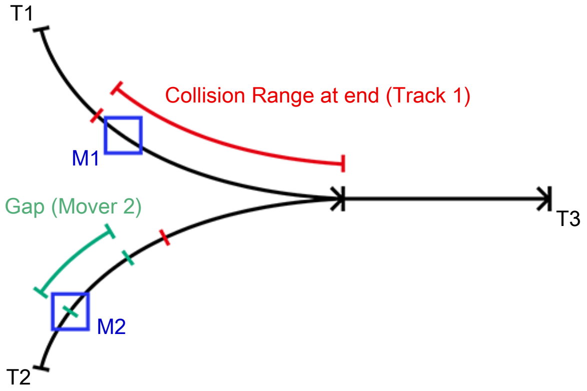
Examples and illustrations:



In this example, the "Collision range at end" for track 1 can be set to zero, because, although two other tracks start at the switch, no other tracks end. The parameter "Collision range at start" for tracks 2 and 3 should be set so that a mover with this distance to the switch cannot collide with movers on the respective other track.

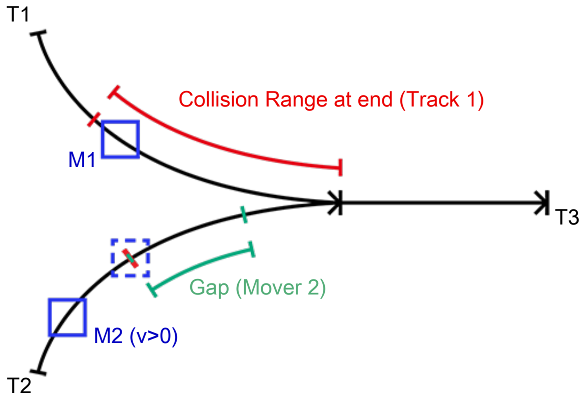


Example of the determination of meaningful Collision Range parameters (T1, T2 and T3 end at the start of T4): If R is the maximum mover radius of movers on the track, a "hose" with radius 2*R can be placed around a track (in this case around track 2) in order to determine a minimum for the Collision Ranges on the other tracks. In this example, track 1 has a smaller "Collision range at end" as it quickly moves away from the other tracks and track 3 and track 2 have a larger "Collision range at end" as they run close together for longer.

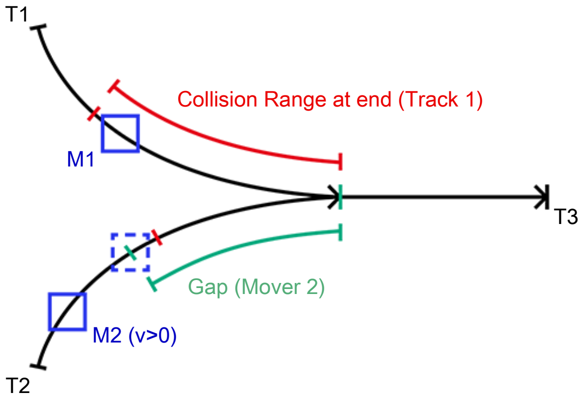


In this example, the additional collision avoidance at the switch is active for Mover 1, since its distance to the switch alone is smaller than the set Collision Range.

Mover 2 is standing still in this example and is further away from the switch than Gap or Collision Range. The additional collision avoidance is therefore not active and the two movers do not have to take each other into consideration at this time.

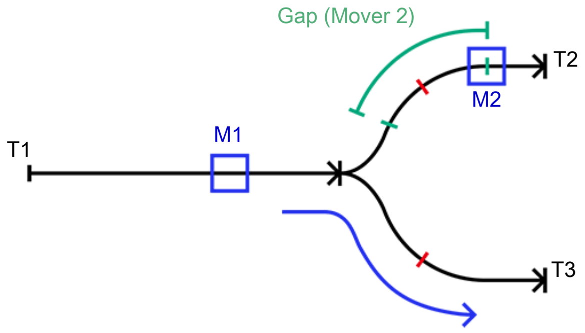


In this example, mover 1 is on track 1 outside the Collision Range, so mover 1 blocks the movement of mover 2 to track 3. Mover 2 stops exactly at the start of the Collision Range of track 2, as this is the last safe stopping point. If the gap between mover 1 and mover 2 would allow it, mover 2 would move to a correspondingly later stop.

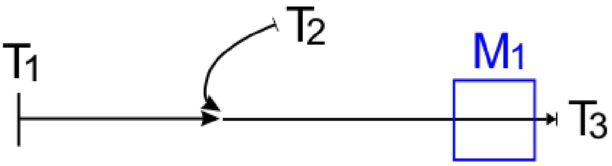


In this example, mover 1 is on track 1 outside the Collision Range, so mover 1 blocks the movement of mover 2 to track 3. Mover 2 stops exactly at the distance of its gap to the end of track 2. If the gap between mover 1 and mover 2 would allow it, mover 2 would move to a correspondingly later stop.

In the last two examples, mover 2 moves on when mover 1 has moved so far forward that both movers have a minimum distance that is greater than the gap between the two movers.



In this example, mover 2 is further away from the switch than the Gap or Collision Range, so mover 1 can drive onto track 3 unhindered. If mover 2 moves back, a blockage may occur if the distance to the switch is less than the Gap or Collision Range.



This is an example of a design to be avoided where the end of a track (in this case T2) affects the Collision Range at the *start* of another track (T3) (and vice versa). In the case of *Automatic Collision Range Mode*, such a situation is not detected. If it is still desired, however, a manual adjustment of the Collision Ranges is necessary here. However, Tracks with such tight curves as T2 in this example are also strongly discouraged due to the strong limitation of the dynamics (tight curves generate large centrifugal forces even when driving through at low velocities).

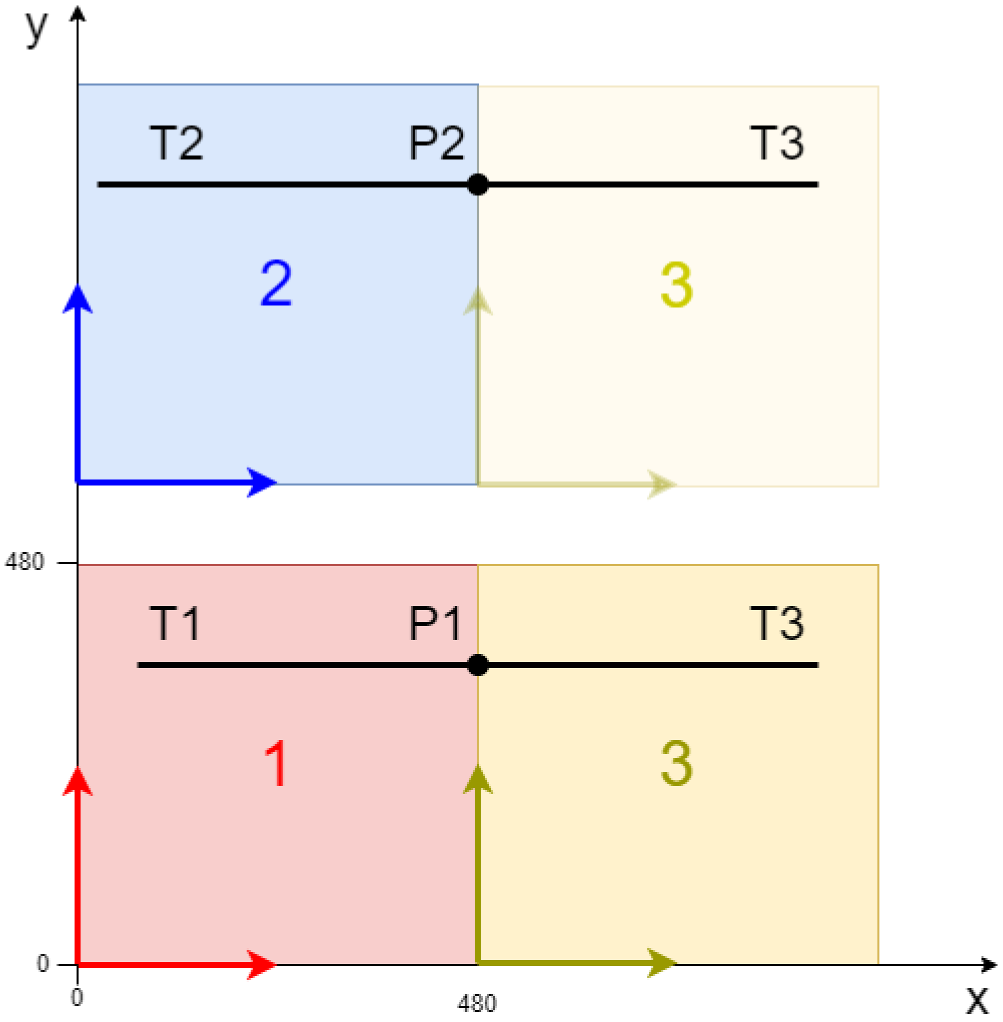
6.2.3 Tracks and parts

From version V3.2.60: The Part feature, which is the subject of this section, is available.

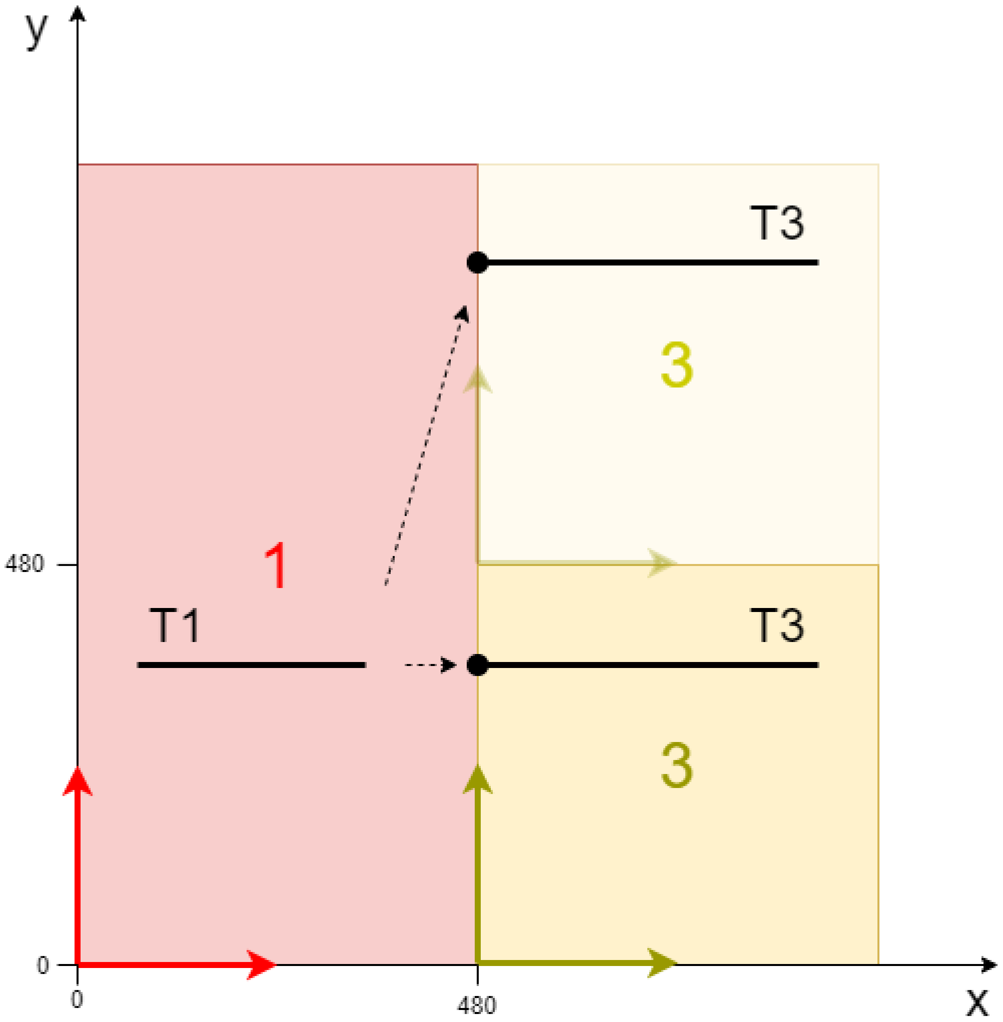
Tracks can be used together with parts, but there are a few special features to bear in mind:

- A track is always permanently assigned to a single part. This is done via the initial parameter "PartOID".
- The track must be geometrically complete on the corresponding part.
- The track has a fixed static geometry relative to its part. So if the position of his part changes, the position of the track changes accordingly.
- If more than one part exists in the configuration, the initial parameter "PartOID" must be specified, otherwise (each) track is automatically assigned to the only part.
- Tracks on the same part can be easily connected. The start and end of each track can only be connected to another track once. This means that StartFromTrack [▶ 165] and EndAtTrack [▶ 165] can be successfully called up a maximum of once per track.
- Tracks on different parts can only be connected if the connection between them lies exactly on the boundary of both parts. This means that the start or end of one of the two tracks must already be on the part boundary so that the other can be connected. This connection can only be crossed if both parts are at this point. If one of the parts is moved to a different position, the connection can no longer be crossed (like any open end of a track). In this case, however, both tracks can be connected to other tracks on other parts in other positions.

To close all these different connections between tracks, the methods StartFromTrack [▶ 165] and EndAtTrack [▶ 165] can be called more than once for each track.



In this example, parts 1 and 2 are static. Part 3 has 2 positions, so that it is connected once to part 1 and once to part 2. Once tracks 1 and 2 have been defined on parts 1 and 2, track 3 on part 3 can be connected to both. Depending on the direction of the tracks, either track 3 must be connected to positions 1 and 2 (or tracks 1 and 2) with two StartFromTrack [▶ 165] calls or with two EndAtTrack [▶ 165] calls. The first call defines the geometry and a logical connection of the start or end of track 3, while the second call only defines a logical connection and only assumes (and checks) that the geometric connection is suitable.



In this example, part 1 is static and part 3 has 2 positions so that it touches part 1 at different points. After track 3 is defined, it is unclear how to interpret a EndAtTrack [▶ 165] from track 1 to track 3. Should track 1 be geometrically connected to track 3 in the upper or lower configuration? This can be realized with the new methods EndAtTrackAdvanced [▶ 167] and StartFromTrackAdvanced [▶ 166]. The exact position of the two parts of the tracks involved is specified in order to connect the tracks in this configuration.

6.2.4 Example "Joining and moving a Planar mover on the track"

Using this guide, you will to create a TwinCAT project that contains two Planar movers and one Planar track. Both movers are joined and moved on the track.

Creating a Planar mover

✓ See Configuration [▶ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
3. Change the start position of the second mover to x = 240.

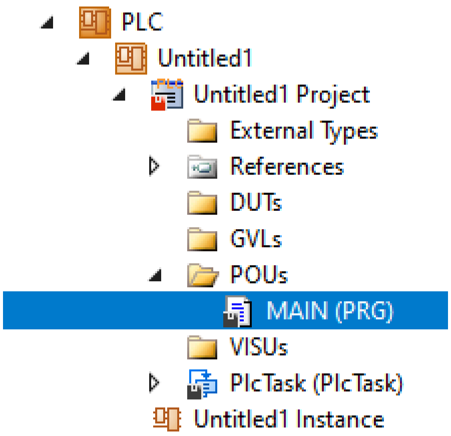
Creating a Planar track

4. Add the Planar track via **Groups > Add New Item…**, see Configuration [▶ 42].

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



⇨ These represent movers and tracks in the MC Configuration.

2. Create two Planar movers, a Planar track, a state variable for a state machine and two auxiliary positions for the track, as shown below.

```
PROGRAM MAIN
VAR
    mover_one, mover_two : MC_PlanarMover;
    track : MC_PlanarTrack;
    state : UDINT;
    pos1, pos2 : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code creates and activates a track and both movers. After that, both movers are joined and moved on the track.

```
CASE state OF
  0:
    pos1.SetValuesXY(0, 0);
    pos2.SetValuesXY(400, 0);
    track.AppendLine(0, pos1, pos2);
    track.Enable(0);
    state := 1;
  1:
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 3;
  3:
    IF mover_one.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled
    AND mover_two.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
  4:
    mover_one.JoinTrack(0, track, 0, 0);
    mover_two.JoinTrack(0, track, 0, 0);
    state := 5;
  5:
    IF mover_one.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack
    AND mover_two.MCTOPLC.STD.CommandMode=MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := 6;
    END_IF
  6:
    mover_one.MoveOnTrack(0, 0, 150.0, 0, 0);
    mover_two.MoveOnTrack(0, 0, 350.0, 0, 0);
    state := 7;
  7:
    IF mover_one.MCTOPLC.SETONTRACK.SetPos >= 149.9
    AND mover_two.MCTOPLC.SETONTRACK.SetPos >= 349.9 THEN
      state := 8;
    END_IF

END_CASE
```

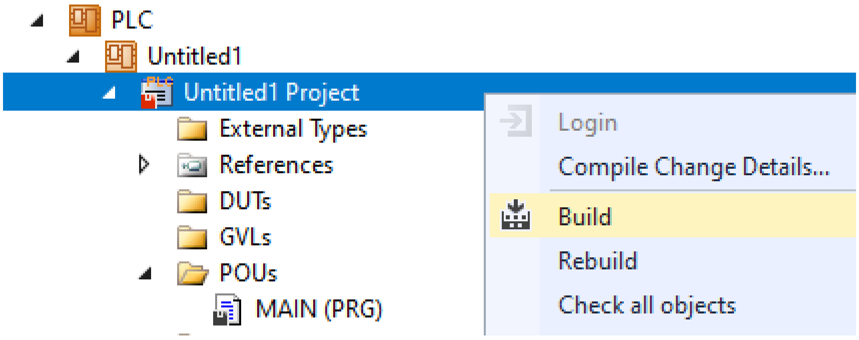
Sending the command

4. To send the command, you must call the movers and the track cyclically with their update method after the END_CASE:

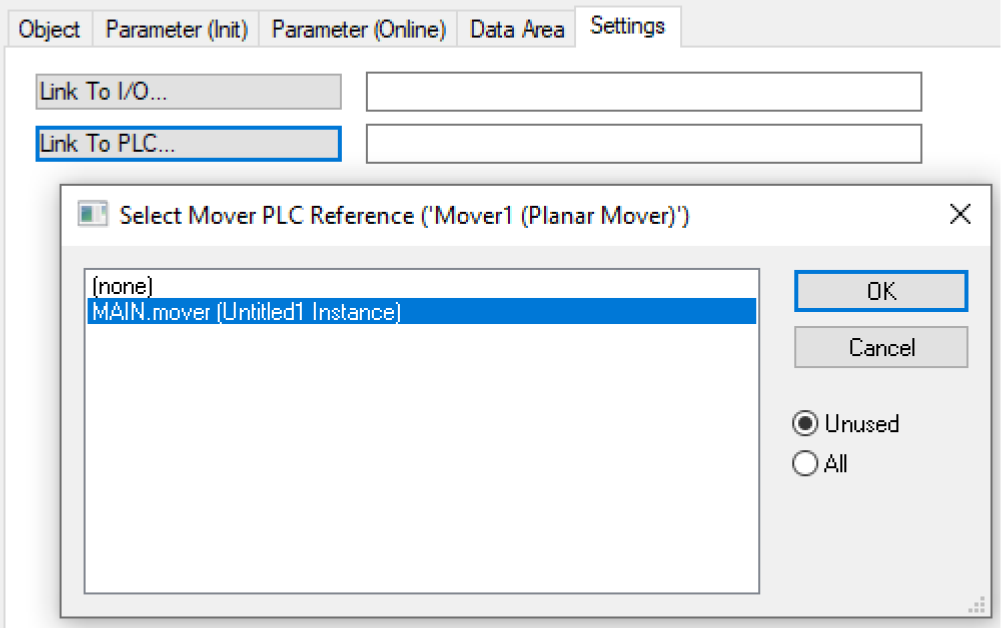
```
mover_one.Update();
mover_two.Update();
track.Update();
```

Building the PLC creates symbols of the "PLC mover" and “track”, which can then be linked to the mover and track instance in the MC project.

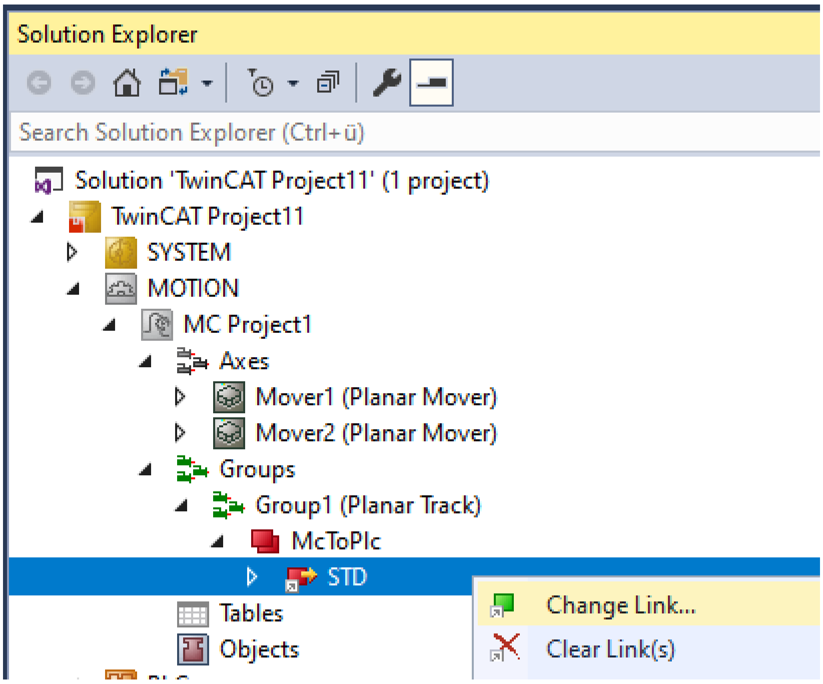
1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**

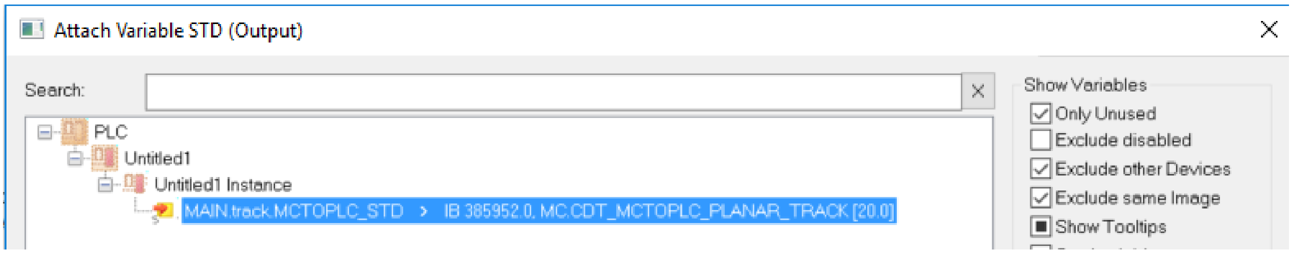


⇨ Subsequently, the Planar movers in the "MC Project" can be linked with the **Link To PLC...** button on the **Settings** tab.



⇨ The track must be linked separately via the following dialog boxes.



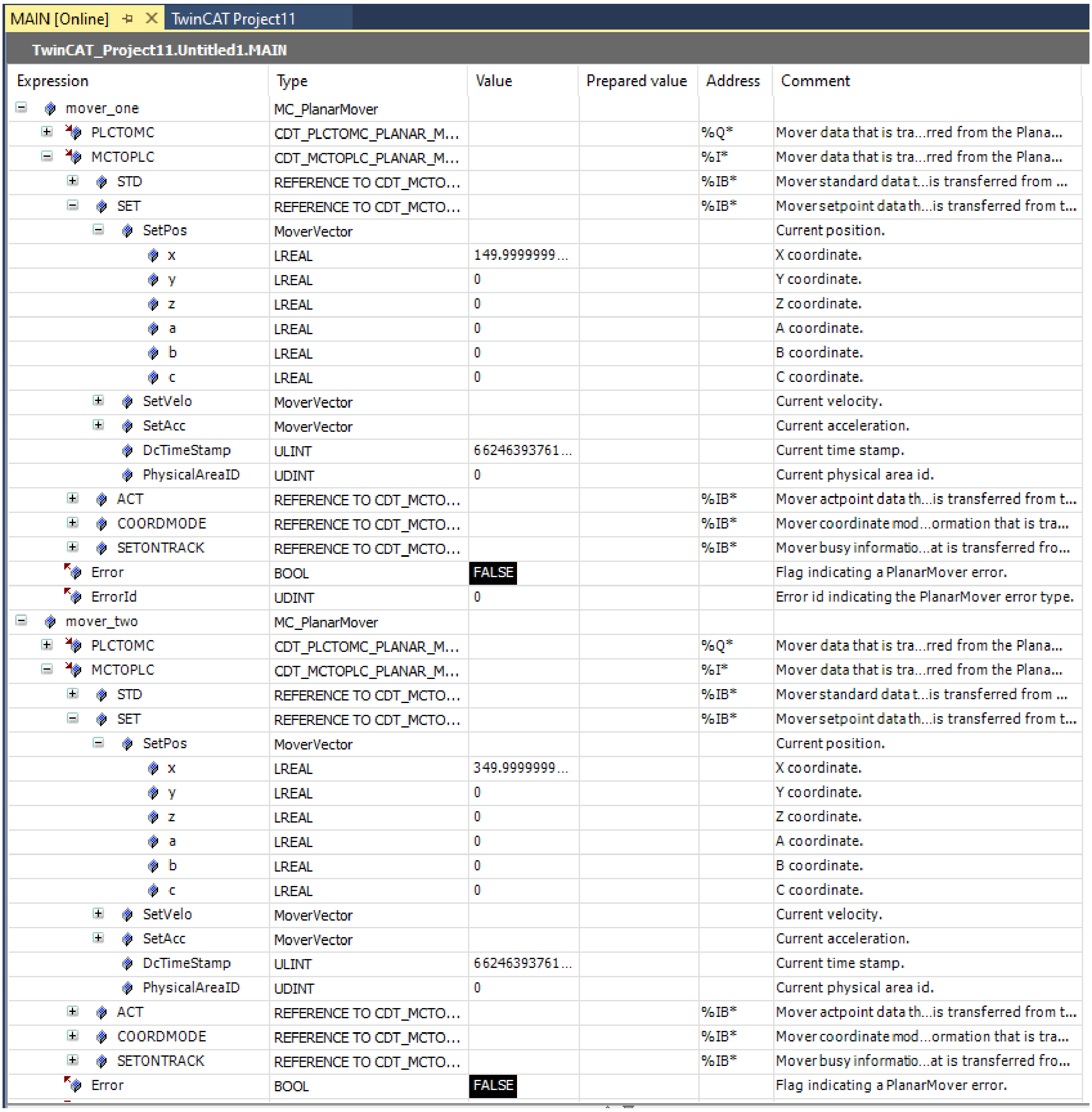

Attach Variable STD (Output)
Search:
PLC
Untitled1
Untitled1 Instance
MAIN.track.MCTOPLC_STD > IB 385952.0, MC.CDT_MCTOPLC_PLANAR_TRACK [20.0]
Show Variables

☑ Only Unused
□ Exclude disabled
☑ Exclude other Devices
☑ Exclude same Image
■ Show Tooltips

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=8), the movers are in the desired positions.



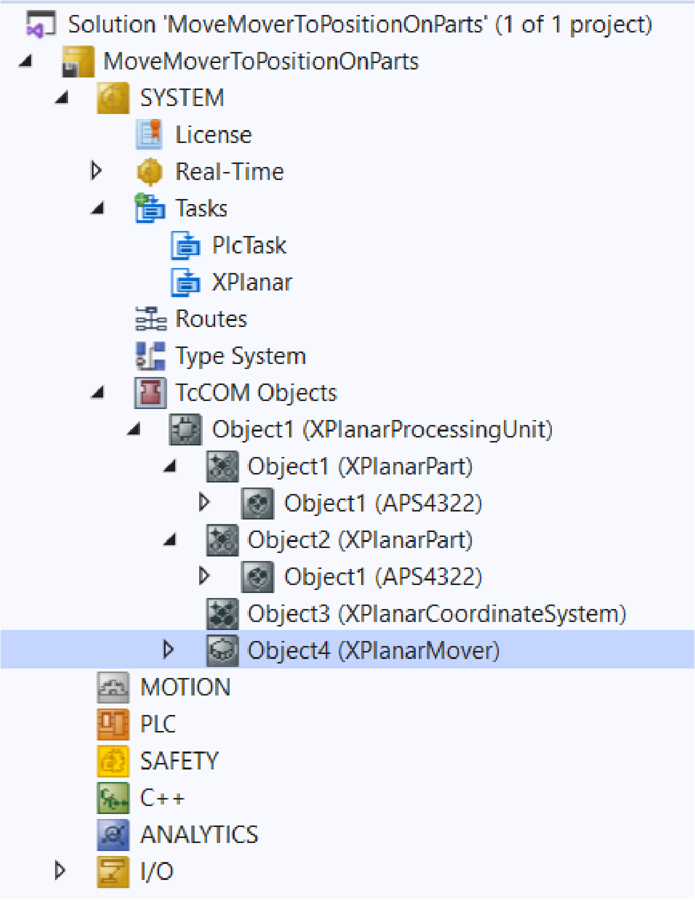
Expression	Type	Value	Prepared value	Address	Comment
⊟ mover_one	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLANAR_M...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLANAR_M...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT_MCTO...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT_MCTO...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	149.9999999...			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	66246393761...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT_MCTO...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT_MCTO...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT_MCTO...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
⊟ mover_two	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLANAR_M...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLANAR_M...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT_MCTO...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT_MCTO...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	349.9999999...			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	66246393761...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT_MCTO...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT_MCTO...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT_MCTO...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.

6.2.5 Example "Moving Planar movers on tracks with Planar parts"

In this example, a Planar mover is moved on two Planar tracks over two Planar parts.

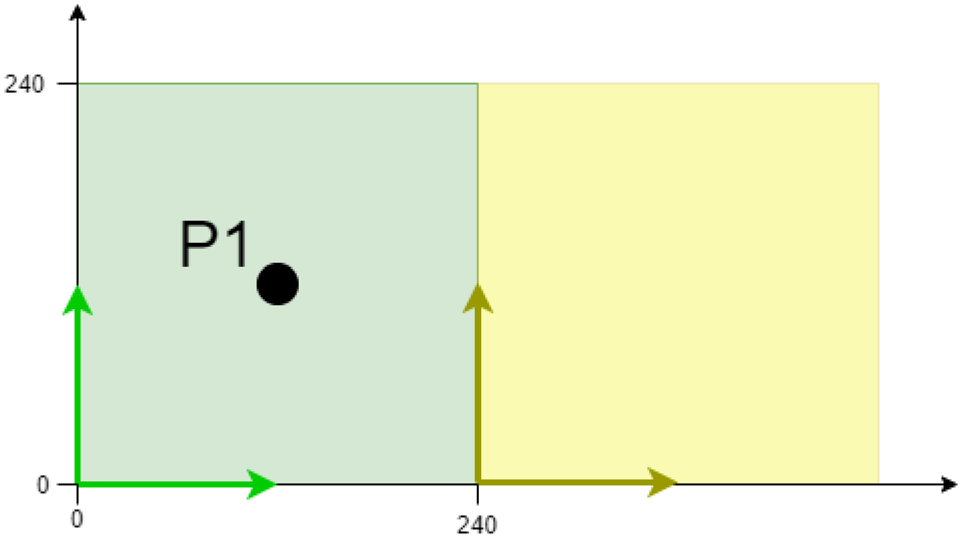
Starting point

You start with a solution that contains a fully configured XPlanar Processing Unit. Two parts, a coordinate system and a mover are created under the XPlanar Processing Unit. A tile is created under each of the two parts.



Solution 'MoveMoverToPositionOnParts' (1 of 1 project)
- MoveMoverToPositionOnParts
  - SYSTEM
    - License
    - Real-Time
    - Tasks
      - PlcTask
      - XPlanar
    - Routes
    - Type System
    - TcCOM Objects
      - Object1 (XPlanarProcessingUnit)
        - Object1 (XPlanarPart)
          - Object1 (APS4322)
        - Object2 (XPlanarPart)
          - Object1 (APS4322)
        - Object3 (XPlanarCoordinateSystem)
        - Object4 (XPlanarMover)
  - MOTION
  - PLC
  - SAFETY
  - C++
  - ANALYTICS
  - I/O

The following geometric situation is set: the two parts are next to each other and the mover starts in the middle of the left part (position P1). Both parts are not movable and the configuration is therefore static.



The example is developed on the basis of this configuration.

The creation of the initial situation is described in the XPlanar Processing Unit documentation.

Creating Planar movers, Planar tracks and Planar environment

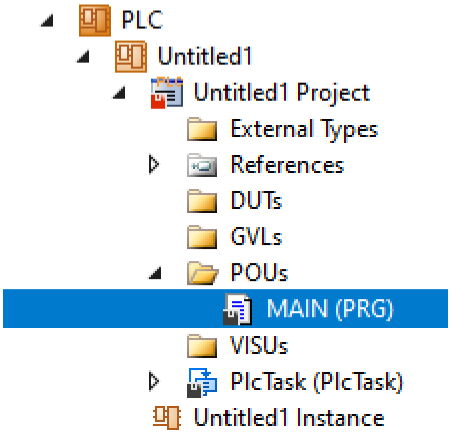
1. Create a Planar mover for this example, see Configuration [▶ 18].
2. Create a Planar environment, see Configuration [▶ 96].

3. Set the initial parameter XPlanar processing unit OID to the object ID of the XPlanar Processing Unit. This activates the Part feature for all **MC Configuration** objects (especially for the created Planar mover).
4. Add two Planar tracks via **Groups > Add New Item…**, see Configuration [▶ 42].
5. Set the initial parameter "PartOID" of the two tracks to the corresponding part; in this example, the first track is set to part 1 and the second track to part 2.

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



⇨ These represent movers and tracks in the MC Configuration.

2. Create a Planar mover, two Planar tracks, a state variable for a state machine and two auxiliary positions for the tracks, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    track_one, track_two : MC_PlanarTrack;
    state : UDINT;
    pos1, pos2 : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code creates and activates two tracks and the mover. The mover is then coupled onto the first track and driven onto the second track, crossing the boundary between Part 1 and Part 2.

```
CASE state OF
  0:
    pos1.SetValuesXYCReferenceId(40, 120, 0, 16#01010060);
    pos2.SetValuesXYCReferenceId(240, 120, 0, 16#01010060);
    track_one.AppendLine(0, pos1, pos2);
    track_two.StartFromTrack(0, track_one);
    pos1.SetValuesXYCReferenceId(260, 120, 0, 16#01010060);
    pos2.SetValuesXYCReferenceId(440, 120, 0, 16#01010060);
    track_two.AppendLine(0, pos1, pos2);
    track_one.Enable(0);
    track_two.Enable(0);
    state := 1;
  1:
    IF track_one.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled AND
    track_two.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    mover.Enable(0);
    state := 3;
  3:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
  4:
    mover.JoinTrack(0, track_one, 0, 0);
    state := 5;
  5:
    IF mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
```

```
      state := 6;
    END_IF
  6:
    mover.MoveOnTrack(0, track_two, 150.0, 0, 0);
    state := 7;

END_CASE
```

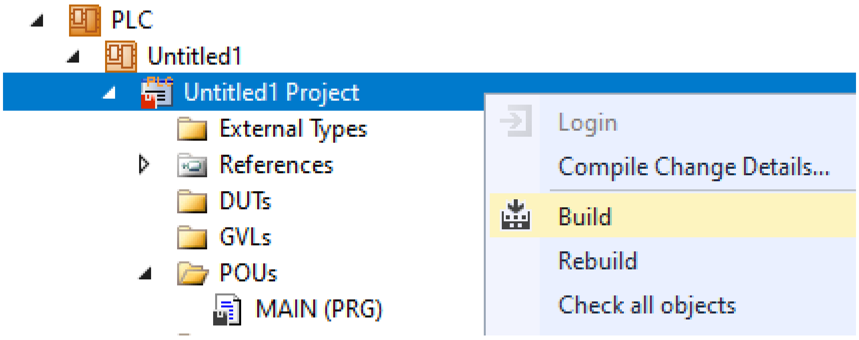
Sending the command

4. To send the command, you must call the movers and the track cyclically with their update method after the END_CASE:

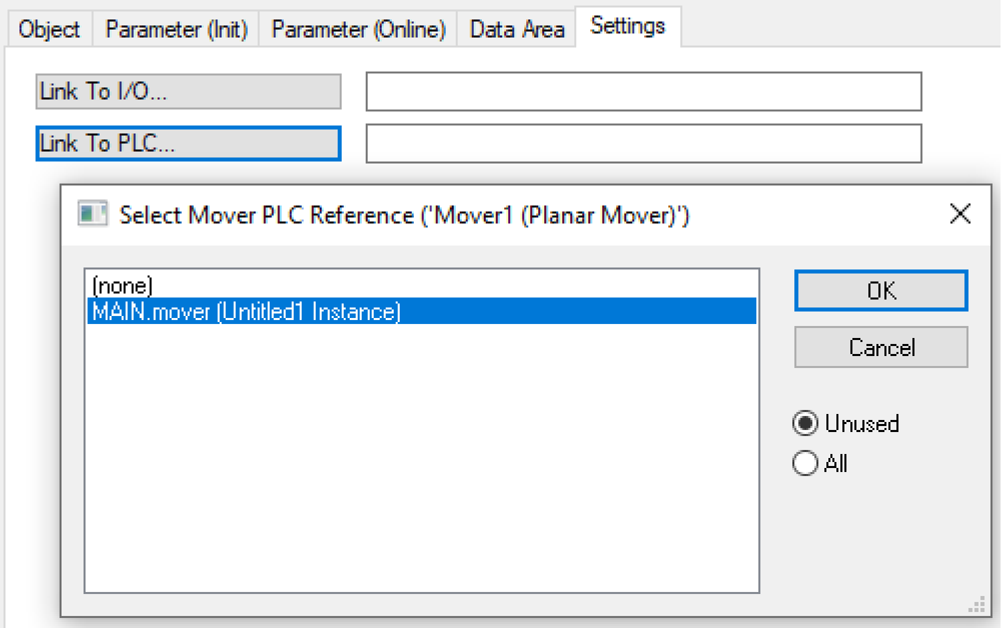
```
mover.Update();
track_one.Update();
track_two.Update();
```

Building the PLC creates symbols of the "PLC mover" and “track”, which can then be linked to the mover and track instance in the MC project.

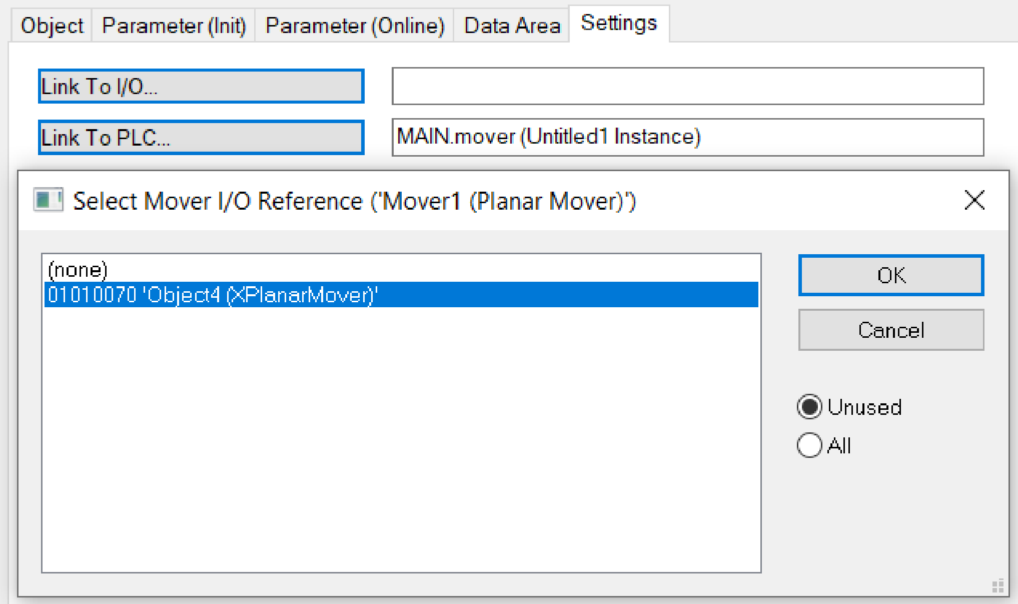
1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ Subsequently, the Planar movers in the "MC Project" can be linked with the **Link To PLC...** button on the **Settings** tab.



⇨ In addition, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To I/O...** button on the **Settings** tab.



Link To I/O...

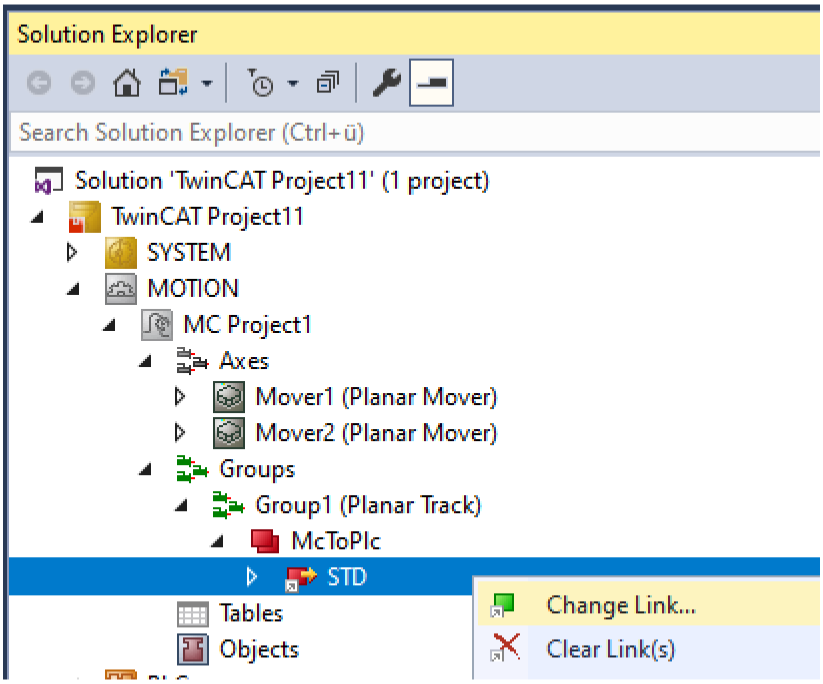
Link To PLC... MAIN.mover (Untitled1 Instance)

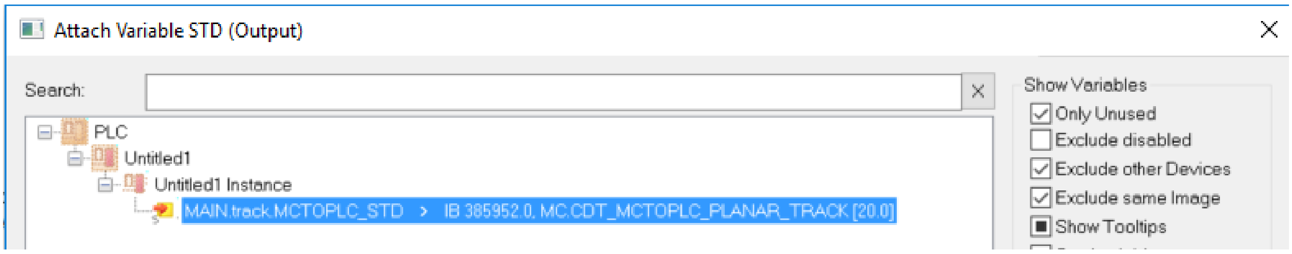
Select Mover I/O Reference ('Mover1 (Planar Mover)')

(none)
01010070 'Object4 (XPlanarMover)'

● Unused
○ All

⇨ The tracks must be linked separately via the following dialog boxes.





Attach Variable STD (Output)

Search:

PLC
Untitled1
Untitled1 Instance
MAIN.track.MCTOPLC_STD > IB 385952.0, MC.CDT_MCTOPLC_PLANAR_TRACK [20.0]

Show Variables
☑ Only Unused
☐ Exclude disabled
☑ Exclude other Devices
☑ Exclude same Image
■ Show Tooltips

Activating and starting the project

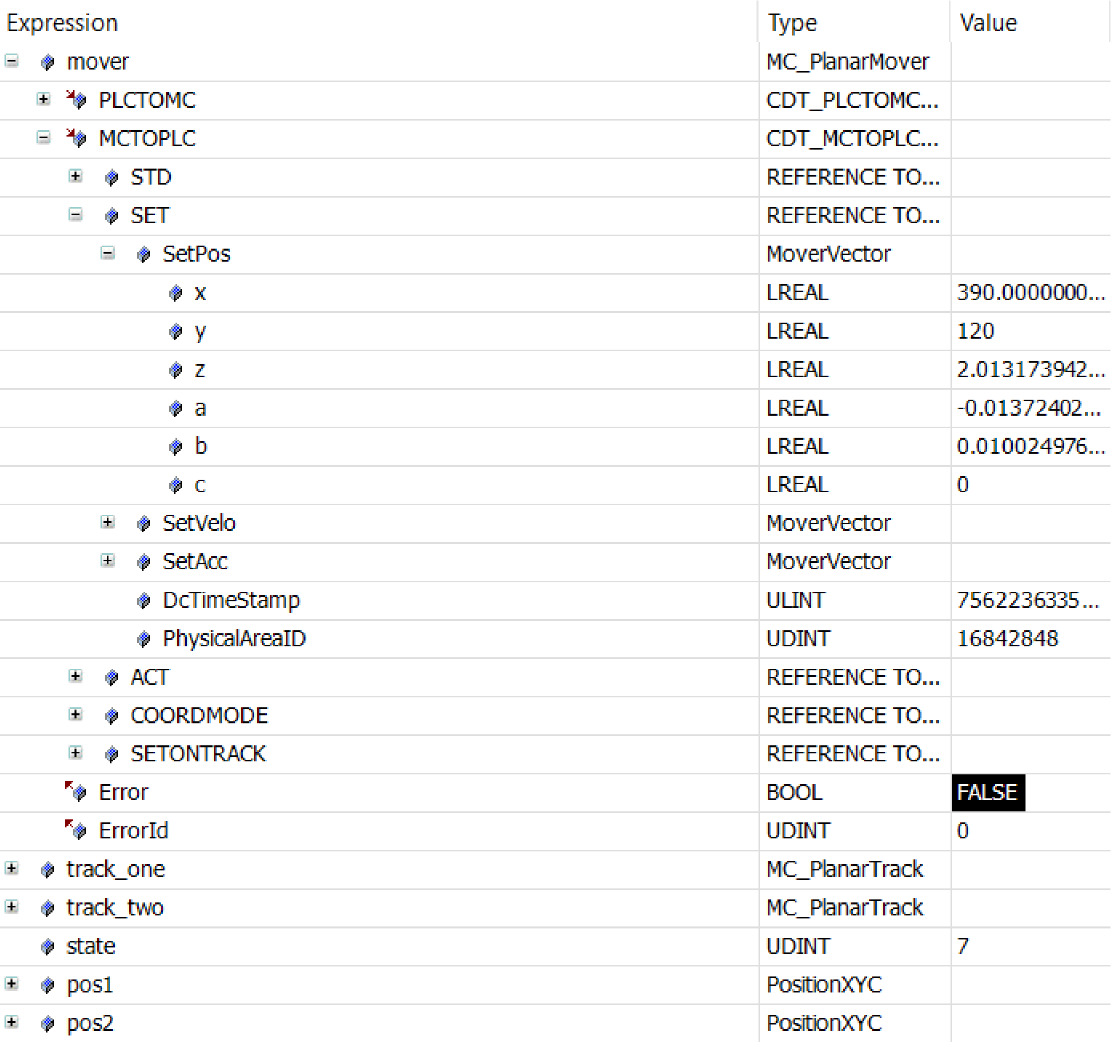
1. Activate the configuration via the button in the menu bar .

2. Set the TwinCAT system to the "Run" state via the button .

3. Log in the PLC via the button in the menu bar .

4. Start the PLC via the Play button in the menu bar.

The mover is at the end of the state machine (state=7) on the second track on part two. The positions of the AppendLine commands were specified in the global coordinate system, as was the end position of the mover.



Expression	Type	Value
mover	MC_PlanarMover	
PLCTOMC	CDT_PLCTOMC...	
MCTOPLC	CDT_MCTOPLC...	
STD	REFERENCE TO...	
SET	REFERENCE TO...	
SetPos	MoverVector	
x	LREAL	390.0000000...
y	LREAL	120
z	LREAL	2.013173942...
a	LREAL	-0.01372402...
b	LREAL	0.010024976...
c	LREAL	0
SetVelo	MoverVector	
SetAcc	MoverVector	
DcTimeStamp	ULINT	7562236335...
PhysicalAreaID	UDINT	16842848
ACT	REFERENCE TO...	
COORDMODE	REFERENCE TO...	
SETONTRACK	REFERENCE TO...	
Error	BOOL	FALSE
ErrorId	UDINT	0
track_one	MC_PlanarTrack	
track_two	MC_PlanarTrack	
state	UDINT	7
pos1	PositionXYC	
pos2	PositionXYC	

6.2.6 Example "Coupling a Planar mover to a track and moving it in CRotationOnTrack mode"

Using this guide, you will to create a TwinCAT project that contains two Planar movers and one Planar track. Both movers are joined and moved on the track.

Creating a Planar mover

✓ See Configuration [▶ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
3. Change the start position of the second mover to x = 240.

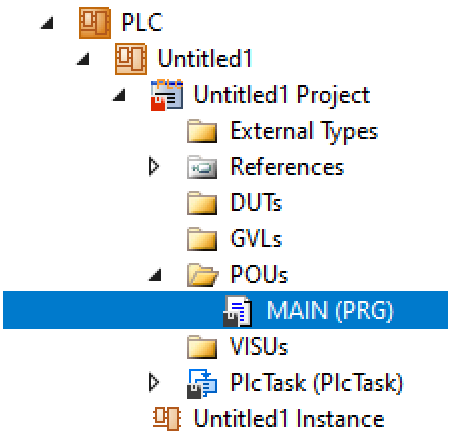
Creating a Planar track

4. Add the Planar track via **Groups > Add New Item…**, see Configuration [▶ 42].

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



```
◢ PLC
  ◢ Untitled1
    ◢ Untitled1 Project
        External Types
      ▷ References
        DUTs
        GVLs
      ◢ POUs
          MAIN (PRG)
        VISUs
      ▷ PlcTask (PlcTask)
      Untitled1 Instance
```

⇨ These represent movers and tracks in the MC Configuration.

2. Create two Planar movers, a Planar track, a state variable for a state machine and two auxiliary positions for the track, as shown below.

```
PROGRAM MAIN
VAR
    mover_one, mover_two : MC_PlanarMover;
    track : MC_PlanarTrack;
    state : UDINT;
    pos1, pos2 : PositionXYC;
    join_track_options : ST_JoinTrackOptions;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code creates and activates a track and both movers. Then both movers are coupled on the track and rotated.

```
CASE state OF
  0:
    pos1.SetValuesXY(0, 0);
    pos2.SetValuesXY(400, 0);
    track.AppendLine(0, pos1, pos2);
    track.Enable(0);
    state := 1;
  1:
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 3;
  3:
    IF mover_one.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled
    AND mover_two.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
  4:
    join_track_options.useOrientation := FALSE;
    mover_one.JoinTrack(0, track, 0, join_track_options);
    mover_two.JoinTrack(0, track, 0, join_track_options);
    state := 5;
  5:
    IF mover_one.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack
    AND mover_two.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := 6;
    END_IF
  6:
    mover_one.MoveC(0, 20.0, 0, 0);
    mover_two.MoveC(0, 90.0, 0, 0);
    state := 7;
  7:
    IF mover_one.MCTOPLC.SET.SetPos.c >= 19.9
    AND mover_two.MCTOPLC.SET.SetPos.c >= 89.9 THEN
      state := 8;
    END_IF

END_CASE
```

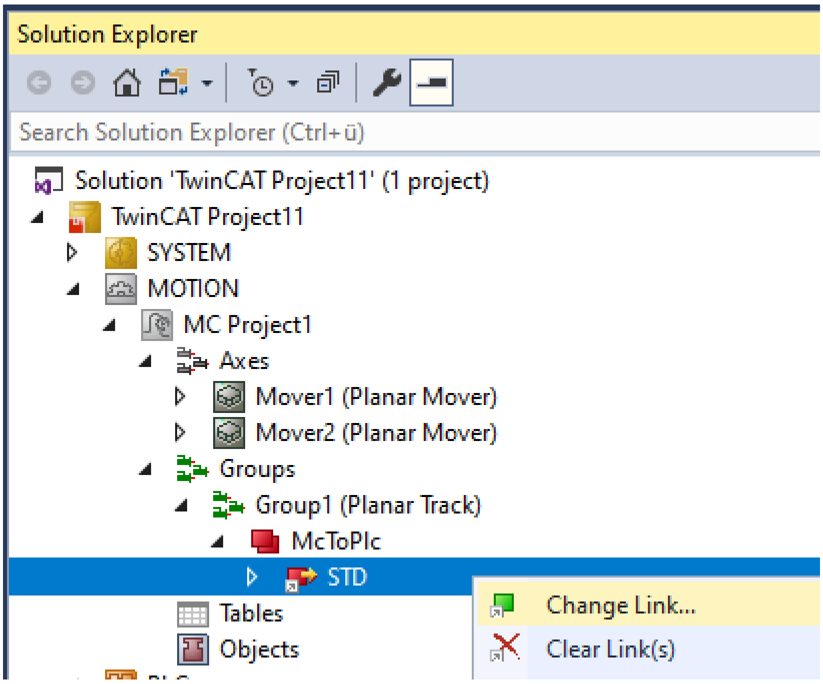
Sending the command

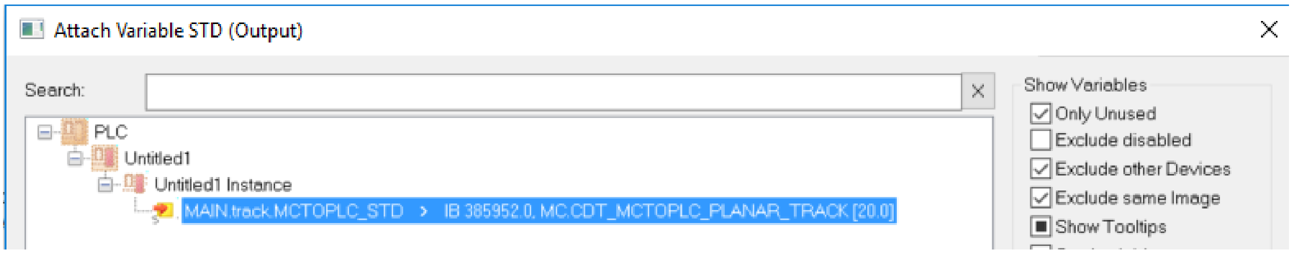
4. To send the command, you must call the movers and the track cyclically with their update method after the END_CASE:

```
mover_one.Update();
mover_two.Update();
track.Update();
```

Building the PLC creates symbols of the "PLC mover" and “track”, which can then be linked to the mover and track instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**
   - ⇨ Subsequently, the Planar movers in the "MC Project" can be linked with the **Link To PLC...** button on the **Settings** tab.
   - ⇨ The track must be linked separately via the following dialog boxes.




Attach Variable STD (Output)

Search:

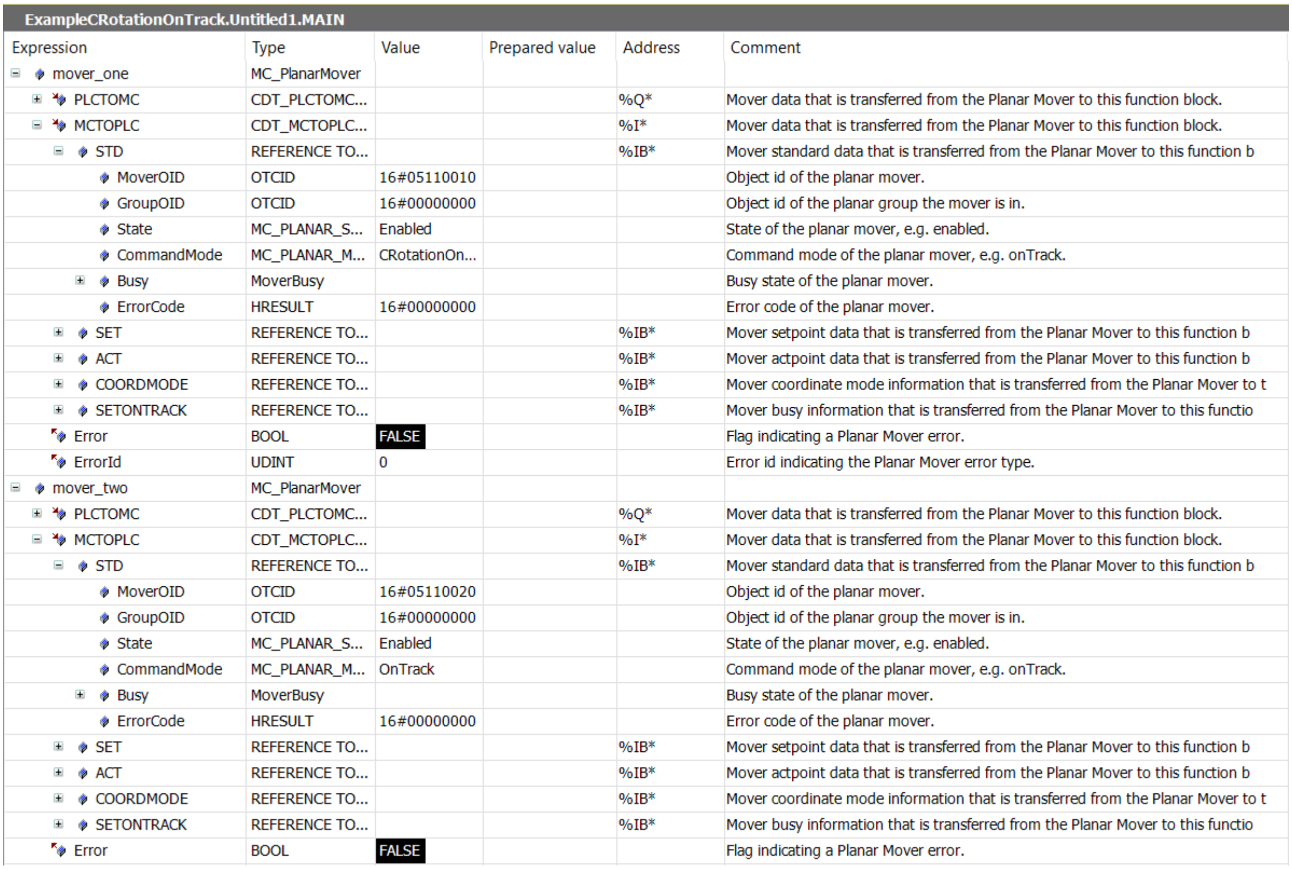
PLC
Untitled1
Untitled1 Instance
MAIN.track.MCTOPLC_STD > IB 385952.0, MC.CDT_MCTOPLC_PLANAR_TRACK [20.0]

Show Variables
☑ Only Unused
☐ Exclude disabled
☑ Exclude other Devices
☑ Exclude same Image
■ Show Tooltips

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=8), the movers are in the desired positions. Mover two is (again) in the OnTrack state and mover one is in the CRotationOnTrack state after both were in the CRotationOnTrack state during the movement. Mover one can now only continue to rotate, while mover two can continue to move on the track or even leave the track.



ExampleCRotationOnTrack.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
⊟ mover_one	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC...			%Q*	Mover data that is transferred from the Planar Mover to this function block.
⊟ MCTOPLC	CDT_MCTOPLC...			%I*	Mover data that is transferred from the Planar Mover to this function block.
⊟ STD	REFERENCE TO...			%IB*	Mover standard data that is transferred from the Planar Mover to this function b
MoverOID	OTCID	16#05110010			Object id of the planar mover.
GroupOID	OTCID	16#00000000			Object id of the planar group the mover is in.
State	MC_PLANAR_S...	Enabled			State of the planar mover, e.g. enabled.
CommandMode	MC_PLANAR_M...	CRotationOn...			Command mode of the planar mover, e.g. onTrack.
⊞ Busy	MoverBusy				Busy state of the planar mover.
ErrorCode	HRESULT	16#00000000			Error code of the planar mover.
⊞ SET	REFERENCE TO...			%IB*	Mover setpoint data that is transferred from the Planar Mover to this function b
⊞ ACT	REFERENCE TO...			%IB*	Mover actpoint data that is transferred from the Planar Mover to this function b
⊞ COORDMODE	REFERENCE TO...			%IB*	Mover coordinate mode information that is transferred from the Planar Mover to t
⊞ SETONTRACK	REFERENCE TO...			%IB*	Mover busy information that is transferred from the Planar Mover to this functio
Error	BOOL	FALSE			Flag indicating a Planar Mover error.
ErrorId	UDINT	0			Error id indicating the Planar Mover error type.
⊟ mover_two	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC...			%Q*	Mover data that is transferred from the Planar Mover to this function block.
⊟ MCTOPLC	CDT_MCTOPLC...			%I*	Mover data that is transferred from the Planar Mover to this function block.
⊟ STD	REFERENCE TO...			%IB*	Mover standard data that is transferred from the Planar Mover to this function b
MoverOID	OTCID	16#05110020			Object id of the planar mover.
GroupOID	OTCID	16#00000000			Object id of the planar group the mover is in.
State	MC_PLANAR_S...	Enabled			State of the planar mover, e.g. enabled.
CommandMode	MC_PLANAR_M...	OnTrack			Command mode of the planar mover, e.g. onTrack.
⊞ Busy	MoverBusy				Busy state of the planar mover.
ErrorCode	HRESULT	16#00000000			Error code of the planar mover.
⊞ SET	REFERENCE TO...			%IB*	Mover setpoint data that is transferred from the Planar Mover to this function b
⊞ ACT	REFERENCE TO...			%IB*	Mover actpoint data that is transferred from the Planar Mover to this function b
⊞ COORDMODE	REFERENCE TO...			%IB*	Mover coordinate mode information that is transferred from the Planar Mover to t
⊞ SETONTRACK	REFERENCE TO...			%IB*	Mover busy information that is transferred from the Planar Mover to this functio
Error	BOOL	FALSE			Flag indicating a Planar Mover error.

6.2.7 Example "Coupling a Planar mover to a track and moving it with AdoptTrackOrientation"

Using this guide, you will to create a TwinCAT project that contains two Planar movers and one Planar track. Both movers are joined and moved on the track.

Creating a Planar mover

✓ See Configuration [▶ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
3. Change the start position of the second mover to x = 240.

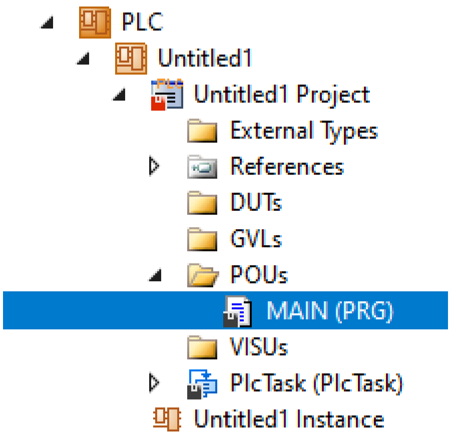
Creating a Planar track

4. Add the Planar track via **Groups > Add New Item…**, see Configuration [▶ 42].

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



- PLC
  - Untitled1
    - Untitled1 Project
      - External Types
      - References
      - DUTs
      - GVLs
      - POUs
        - MAIN (PRG)
      - VISUs
      - PlcTask (PlcTask)
    - Untitled1 Instance

⇨ These represent movers and tracks in the MC Configuration.

2. Create two Planar movers, a Planar track, a state variable for a state machine and two auxiliary positions for the track, as shown below.

```
PROGRAM MAIN
VAR
    mover_one, mover_two : MC_PlanarMover;
    track : MC_PlanarTrack;
    state : UDINT;
    pos1, pos2 : PositionXYC;
    join_track_options : ST_JoinTrackOptions;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code creates and activates a track and both movers. Then both movers are coupled on the track and rotated.

```
CASE state OF
  0:
    pos1.SetValuesXY(0, 0);
    pos2.SetValuesXY(400, 0);
    track.AppendLine(0, pos1, pos2);
    track.Enable(0);
    state := 1;
  1:
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 3;
  3:
    IF mover_one.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled
    AND mover_two.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
  4:
    join_track_options.useOrientation := TRUE;
    mover_one.JoinTrack(0, track, 0, join_track_options);
    mover_two.JoinTrack(0, track, 0, join_track_options);
    state := 5;
  5:
    IF mover_one.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack
    AND mover_two.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := 6;
    END_IF
  6:
    mover_one.MoveC(0, 20.0, 0, 0);
    mover_two.MoveC(0, 190.0, 0, 0);
    state := 7;
  7:
    IF mover_one.MCTOPLC.SET.SetPos.c >= 19.9
    AND NOT mover_one.MCTOPLC.STD.Busy.busyMover
    AND mover_two.MCTOPLC.SET.SetPos.c >= 189.9
    AND NOT mover_two.MCTOPLC.STD.Busy.busyMover THEN
      state := 8;
    END_IF
  8:
    mover_one.AdoptTrackOrientation(0, 0, 0);
    mover_two.AdoptTrackOrientation (0, 0, 0);
    state := 9;

END_CASE
```

Sending the command

4. To send the command, you must call the movers and the track cyclically with their update method after the END_CASE:

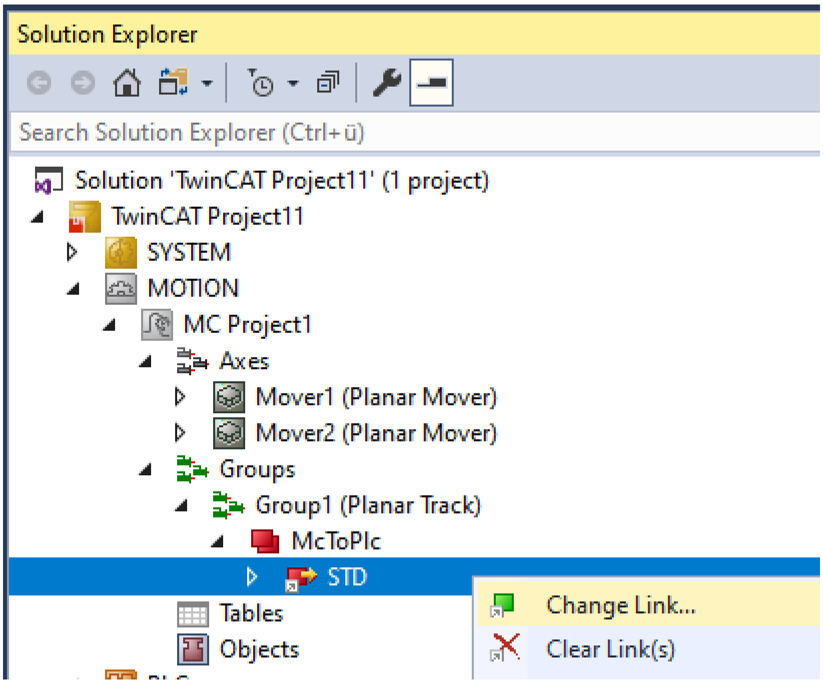
```
mover_one.Update();
mover_two.Update();
track.Update();
```

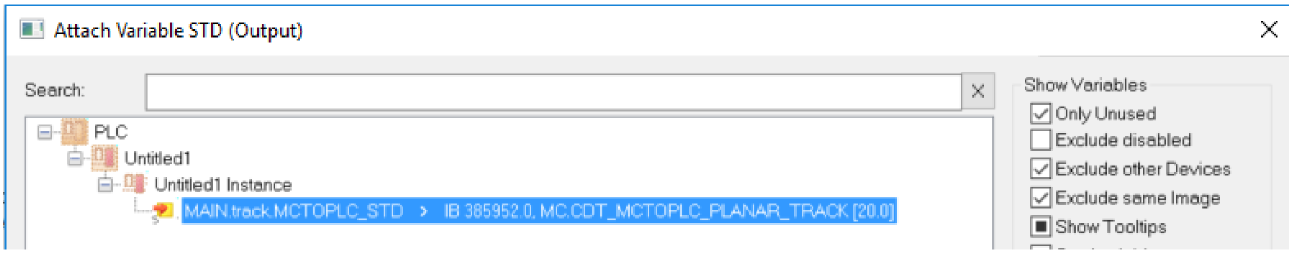
Building the PLC creates symbols of the "PLC mover" and “track”, which can then be linked to the mover and track instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**

   ⇨ Subsequently, the Planar movers in the "MC Project" can be linked with the **Link To PLC...** button on the **Settings** tab.

   ⇨ The track must be linked separately via the following dialog boxes.





MAIN.track.MCTOPLC_STD > IB 385952.0, MC.CDT_MCTOPLC_PLANAR_TRACK [20.0]

Show Variables
☑ Only Unused
☐ Exclude disabled
☑ Exclude other Devices
☑ Exclude same Image
■ Show Tooltips

Activating and starting the project

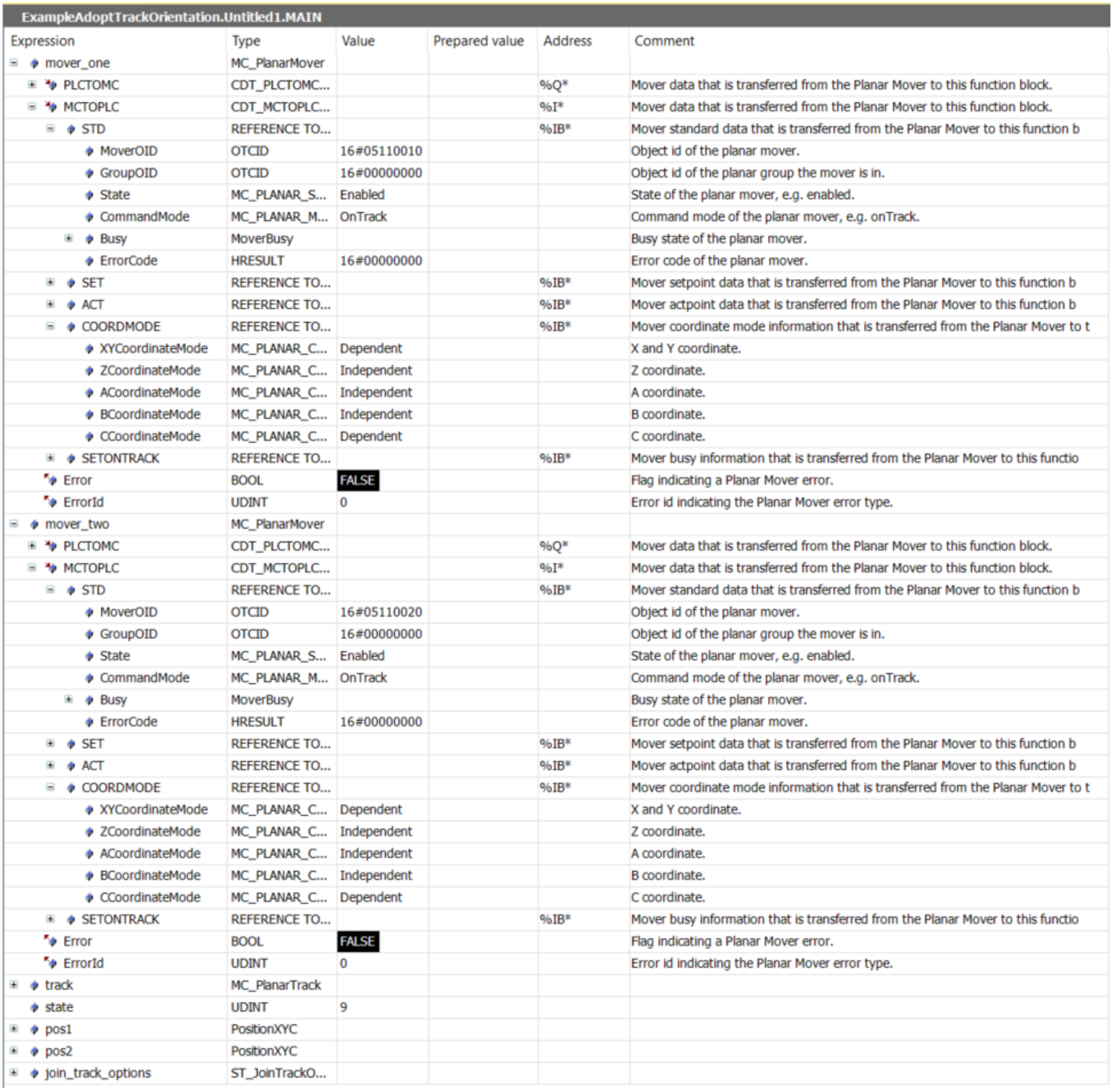
1. Activate the configuration via the button in the menu bar .

2. Set the TwinCAT system to the "Run" state via the button.

3. Log in the PLC via the button in the menu bar .

4. Start the PLC via the Play button in the menu bar.

Both movers are added to the track with orientation coupled to the track. Afterwards the orientation is decoupled from the track by a MoveC. At the end of the state machine (state=9), the movers are in the desired positions. Both movers are (again) in the OnTrack state and have the orientation coupled to the track again by the AdoptTrackOrientation command. The command has 3 parameters: first, an optional Feedback object, second, an optional Constraints object, and third, an optional Options object.



ExampleAdoptTrackOrientation.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
mover_one	MC_PlanarMover				
PLCTOMC	CDT_PLCTOMC...			%Q*	Mover data that is transferred from the Planar Mover to this function block.
MCTOPLC	CDT_MCTOPLC...			%I*	Mover data that is transferred from the Planar Mover to this function block.
STD	REFERENCE TO...			%IB*	Mover standard data that is transferred from the Planar Mover to this function b
MoverOID	OTCID	16#05110010			Object id of the planar mover.
GroupOID	OTCID	16#00000000			Object id of the planar group the mover is in.
State	MC_PLANAR_S...	Enabled			State of the planar mover, e.g. enabled.
CommandMode	MC_PLANAR_M...	OnTrack			Command mode of the planar mover, e.g. onTrack.
Busy	MoverBusy				Busy state of the planar mover.
ErrorCode	HRESULT	16#00000000			Error code of the planar mover.
SET	REFERENCE TO...			%IB*	Mover setpoint data that is transferred from the Planar Mover to this function b
ACT	REFERENCE TO...			%IB*	Mover actpoint data that is transferred from the Planar Mover to this function b
COORDMODE	REFERENCE TO...			%IB*	Mover coordinate mode information that is transferred from the Planar Mover to t
XYCoordinateMode	MC_PLANAR_C...	Dependent			X and Y coordinate.
ZCoordinateMode	MC_PLANAR_C...	Independent			Z coordinate.
ACoordinateMode	MC_PLANAR_C...	Independent			A coordinate.
BCoordinateMode	MC_PLANAR_C...	Independent			B coordinate.
CCoordinateMode	MC_PLANAR_C...	Dependent			C coordinate.
SETONTRACK	REFERENCE TO...			%IB*	Mover busy information that is transferred from the Planar Mover to this functio
Error	BOOL	FALSE			Flag indicating a Planar Mover error.
ErrorId	UDINT	0			Error id indicating the Planar Mover error type.
mover_two	MC_PlanarMover				
PLCTOMC	CDT_PLCTOMC...			%Q*	Mover data that is transferred from the Planar Mover to this function block.
MCTOPLC	CDT_MCTOPLC...			%I*	Mover data that is transferred from the Planar Mover to this function block.
STD	REFERENCE TO...			%IB*	Mover standard data that is transferred from the Planar Mover to this function b
MoverOID	OTCID	16#05110020			Object id of the planar mover.
GroupOID	OTCID	16#00000000			Object id of the planar group the mover is in.
State	MC_PLANAR_S...	Enabled			State of the planar mover, e.g. enabled.
CommandMode	MC_PLANAR_M...	OnTrack			Command mode of the planar mover, e.g. onTrack.
Busy	MoverBusy				Busy state of the planar mover.
ErrorCode	HRESULT	16#00000000			Error code of the planar mover.
SET	REFERENCE TO...			%IB*	Mover setpoint data that is transferred from the Planar Mover to this function b
ACT	REFERENCE TO...			%IB*	Mover actpoint data that is transferred from the Planar Mover to this function b
COORDMODE	REFERENCE TO...			%IB*	Mover coordinate mode information that is transferred from the Planar Mover to t
XYCoordinateMode	MC_PLANAR_C...	Dependent			X and Y coordinate.
ZCoordinateMode	MC_PLANAR_C...	Independent			Z coordinate.
ACoordinateMode	MC_PLANAR_C...	Independent			A coordinate.
BCoordinateMode	MC_PLANAR_C...	Independent			B coordinate.
CCoordinateMode	MC_PLANAR_C...	Dependent			C coordinate.
SETONTRACK	REFERENCE TO...			%IB*	Mover busy information that is transferred from the Planar Mover to this functio
Error	BOOL	FALSE			Flag indicating a Planar Mover error.
ErrorId	UDINT	0			Error id indicating the Planar Mover error type.
track	MC_PlanarTrack				
state	UDINT	9			
pos1	PositionXYC				
pos2	PositionXYC				
join_track_options	ST_JoinTrackO...				

6.2.8 Example "Synchronizing a Planar mover on a track with one axis"

Using these instructions, you will create a TwinCAT project in which a Planar mover located on a track is coupled to an axis whose setpoints it then follows.

In this case, the Planar mover is not controlled directly by a MoveOnTrack command, in which a specified target position is approached with subsequent halt, see Example "Joining and moving a Planar mover on the track" [▶ 50]. Instead, the Planar mover remains coupled to an axis until a subsequent command terminates this coupling, or an error occurs.

After sending the GearInPosOnTrack [▶ 147] command that initiates the coupling to an axis, the Planar mover will attempt to be at the specified slaveSyncPosition if the axis it is coupled to is at the masterSyncPosition and simultaneously assumes the dynamics of the master axis. If synchronicity can be reached earlier (i.e. the Planar mover already has the same dynamics at slaveSyncPosition – x as the master axis, which is at masterSyncPosition – x at this time), then the Planar mover will activate this configuration and become synchronous earlier. If synchronicity cannot be reached at the specified time, the Planar mover will attempt to synchronize with the master axis until a subsequent command is received or an error occurs.

If the Planar mover loses its synchronization status, e.g. due to rapidly changing dynamics of the master axis, it will try to synchronize again as soon as possible. The synchronization status can be accessed at any time from the PLC via the corresponding feedback object. Synchronization can also be lost if maintaining the specified distance from the Planar mover that is ahead requires the synchronous Planar mover to decelerate. Again, the system tries to regain synchronization as quickly as possible once the obstacle is removed.

An example of an error that causes the command to abort is a master axis behavior that would force the Planar mover to move at negative velocity beyond the start of a Planar track. Such a movement is not permitted even with a MoveOnTrack [▶ 146] command. In such a case, the Planar mover will remain in sync (or try to sync, if it isn't already) until it is forced to stop so that it comes to a halt at the beginning of the Planar track. In addition, an error is reported back. The exact position at which the Planar mover initiates its stop depends on the current dynamic limits.

If the GearInPosOnTrack [▶ 147] command is given dynamic limits whose velocity limit is below the current velocity of the master axis, the Planar mover will nevertheless attempt to synchronize, since it cannot be ruled out that the master axis will decelerate at a later point in time in such a way that it can be reached again. In particular, no error is returned in such a case.

Creating a Planar mover

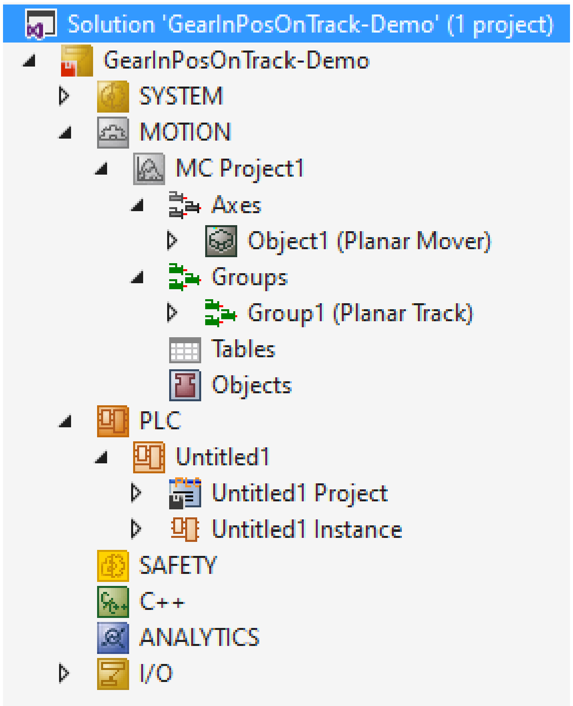
✓ See Configuration [▶ 18].

1. Create a Planar mover.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a Planar track

3. Add the Planar track via **Groups > Add New Item…**, see Configuration [▶ 42].

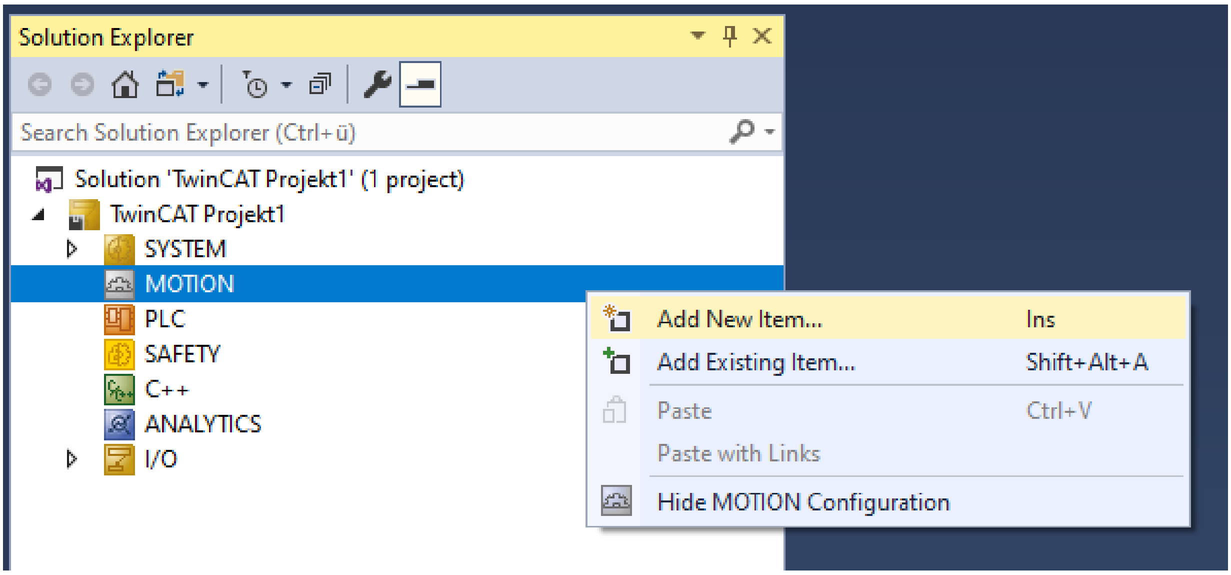
⇨ The Solution Explorer has the following entries:



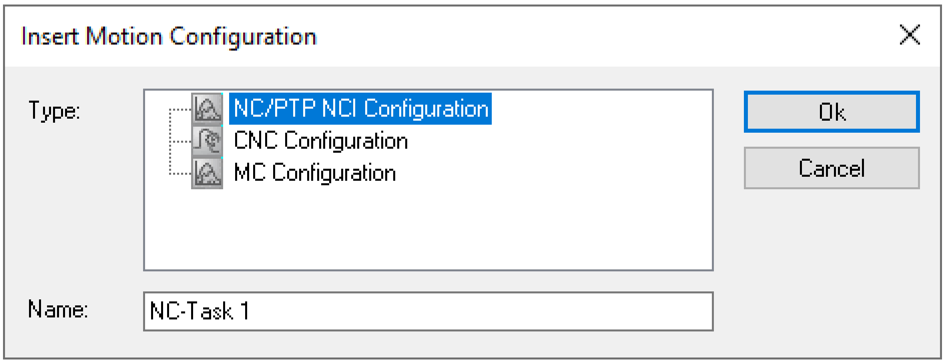
Creating a master axis

✓ To create a master axis, an **NC/PTP NCI configuration** must first be created.

1. Select **MOTION** > **Add New Item…**.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Projekt1' (1 project)
TwinCAT Projekt1
SYSTEM
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
Add New Item... Ins
Add Existing Item... Shift+Alt+A
Paste Ctrl+V
Paste with Links
Hide MOTION Configuration

2. In the following dialog box, select **NC/PTP NCI Configuration** and confirm with **OK**.

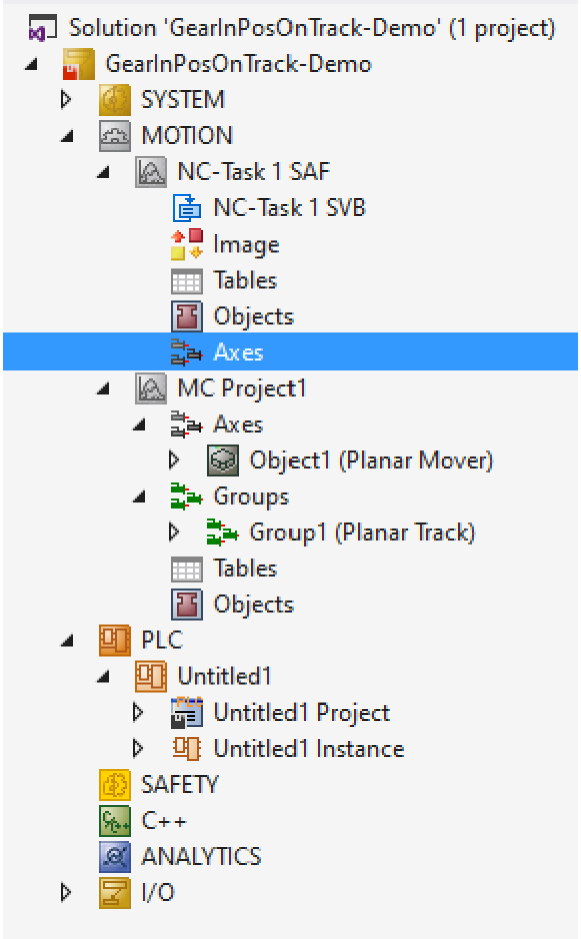

Insert Motion Configuration

Type: NC/PTP NCI Configuration

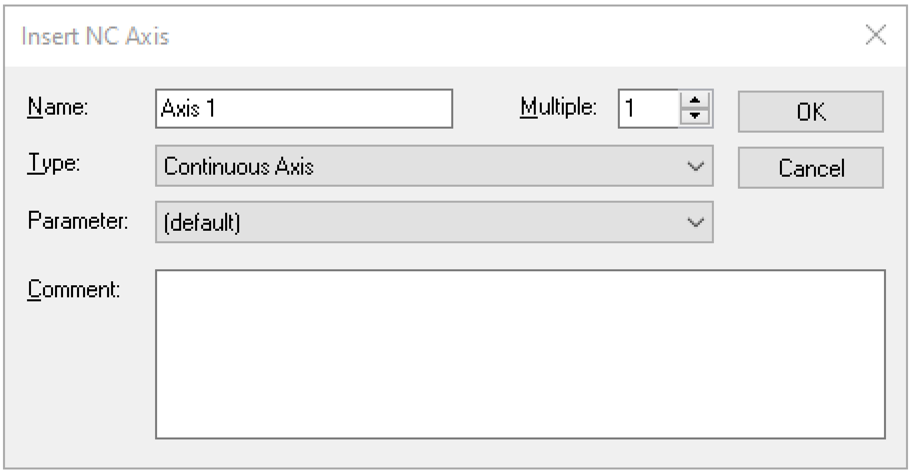
Name: NC-Task 1

⇨ You have created an NC/PTP NCI Project.

3. Right-click in the created NC project **Axes** > **Add New Item...**.



4. In the following dialog box, create one (or more) axes and confirm with **OK**



Insert NC Axis

Name: Axis 1

Multiple: 1

Type: Continuous Axis

Parameter: (default)

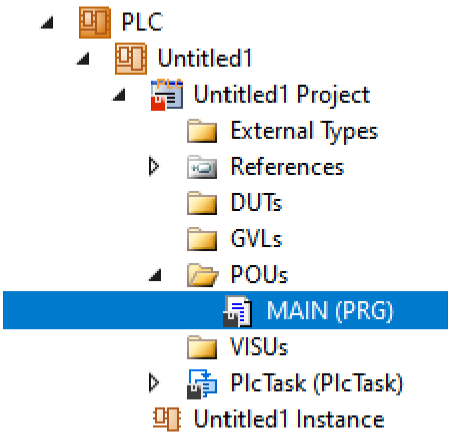
Comment:

Creating a PLC

For this PLC project, you must also add "Tc2_MC2" to control the master axis, see Inserting libraries [▶ 125].

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**

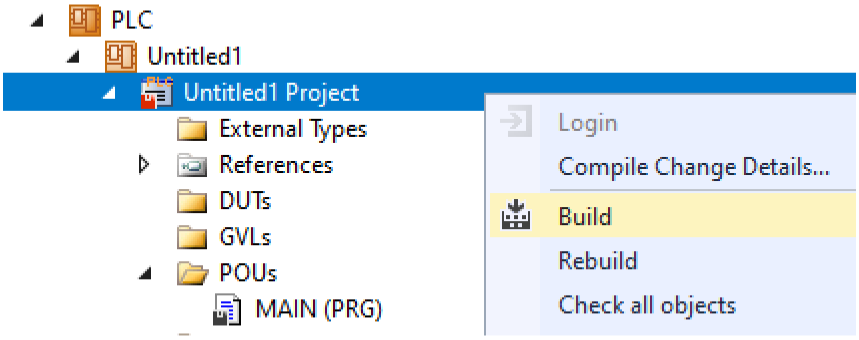


⇨ These represent movers and tracks in the MC Configuration.

2. Create the following variables.

```
PROGRAM MAIN
VAR
    mover       : MC_PlanarMover;
    track       : MC_PlanarTrack;
    axis        : AXIS_REF;
    power_axis  : MC_Power;
    move_axis   : MC_MoveAbsolute;
    state       : UDINT;
    pos1, pos2  : PositionXYC;
END_VAR
```

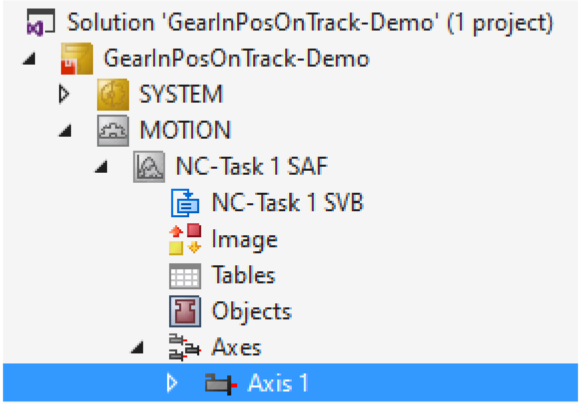
3. Build the PLC to create symbols of the "PLC mover", the "PLC track" and the "PLC axis".



4. Link the Planar mover, Planar track (see Example "Joining and moving a Planar mover on the track" [▶ 50]) and the axis, as described in the next section.

Linking an axis

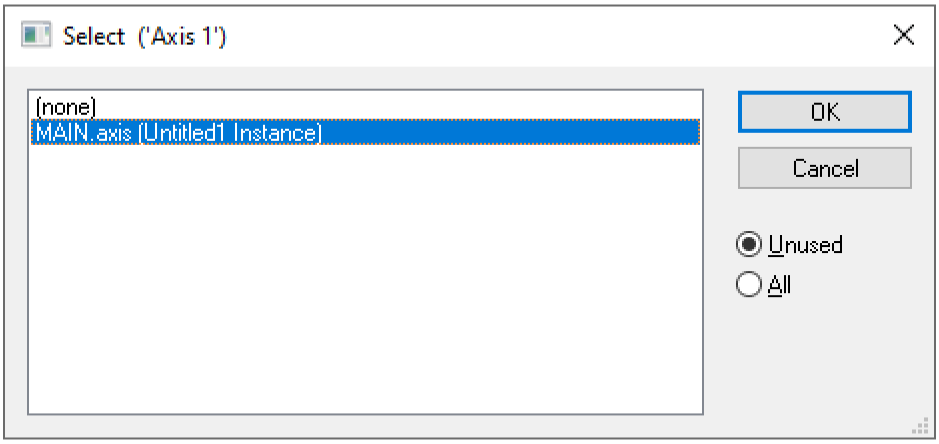
5. Double-click **Axis 1**



in the Solution Explorer.

6. Switch to the **Settings** tab.

7. Click **Link to PLC...** and select in the dialog that follows the entry **MAIN.axis** and confirm with **OK**.



Select ('Axis 1')

(none)
MAIN.axis (Untitled1 Instance)

Programming state machines

With the following state machine, which is programmed in MAIN, the Planar track is geometrically defined and activated (State 0), the Planar mover is activated and coupled to the Planar track (State 2 or 4), and the master axis is enabled (State 6) and moved (State 7).

Finally, the command to start synchronization with the master axis ("GearInPosOnTrack [▶ 147]") is sent to the Planar mover (State 8). Here, too, the Planar objects are updated cyclically or the axis FBs are called (after `END_CASE` statement):

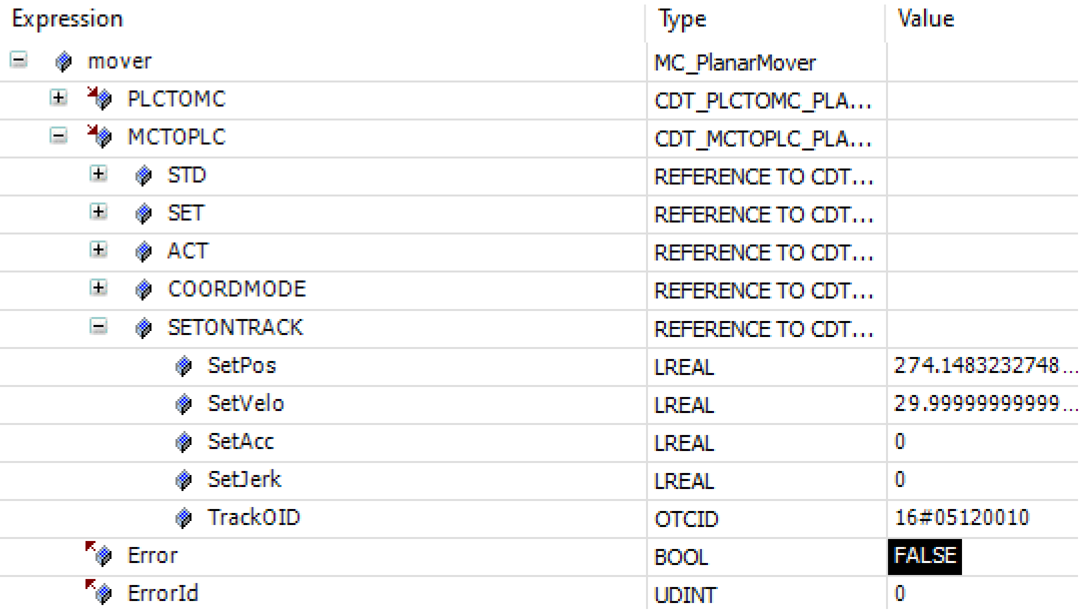
```
CASE state OF
  0:
    pos1.SetValuesXYC(100, 100, 0);
    pos2.SetValuesXYC(860, 100, 0);
    track.AppendLine(0, pos1, pos2);
    track.Enable(0);
    state := state + 1;
  1:
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  2:
    mover.Enable(0);
    state := state + 1;
  3:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  4:
    mover.JoinTrack(0, track, 0, 0);
    state := state + 1;
  5:
    IF mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
    END_IF
  6:
    power_axis(Axis := axis,
    Enable := TRUE,
    Enable_Positive := TRUE);
    IF power_axis.Status THEN
      move_axis(Axis := axis, Execute := FALSE);
      state := state + 1;
    END_IF
  7:
    move_axis(Axis := axis,
    Position := 600,
    Velocity := 30,
    Acceleration := 100,
    Deceleration := 100,
    Jerk := 100,
    Execute := TRUE);
    state := state + 1;
  8:
    mover.GearInPosOnTrack(0, axis.DriveAddress.TcAxisObjectId, 0, 100, 100, track, 0, 0);
    state := state + 1;
END_CASE
```

```
mover.Update();
track.Update();
power_axis(Axis := axis);
move_axis(Axis := axis);
axis.ReadStatus();
```

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

The master axis will move to the given target position (600 in this case), and the Planar mover will follow its movement. The position of the Planar mover can be tracked in the online view (by clicking the button).



Expression	Type	Value
mover	MC_PlanarMover	
PLCTOMC	CDT_PLCTOMC_PLA...	
MCTOPLC	CDT_MCTOPLC_PLA...	
STD	REFERENCE TO CDT...	
SET	REFERENCE TO CDT...	
ACT	REFERENCE TO CDT...	
COORDMODE	REFERENCE TO CDT...	
SETONTRACK	REFERENCE TO CDT...	
SetPos	LREAL	274.1483232748...
SetVelo	LREAL	29.99999999999...
SetAcc	LREAL	0
SetJerk	LREAL	0
TrackOID	OTCID	16#05120010
Error	BOOL	FALSE
ErrorId	UDINT	0

The mover comes to a halt at position 600, since the master axis also reaches zero dynamics here. If a value greater than the length of the track (760 in this case) is programmed in State 7 for the target position of the master axis, the Planar mover comes to a halt at the end of the Planar track in order not to derail and does not follow the master axis any further. The error in such a scene is potentially returned to the PLC by the MC, but is not accepted by the above PLC code in this case. A feedback [▶ 119] object is required for this purpose and for monitoring the synchronization status.

In the function call in State 8, the sync positions of the master axis (third argument) or the Slave Planar Mover (fourth argument) are passed to the Planar mover. These are the respective positions at which the slave becomes synchronous with the master, i.e. at which it reaches its dynamic values. The fifth argument in the function call specifies the Planar track to which the position in the previous argument refers. In fact, it is possible for the slave to get in sync with its master significantly sooner.

A synchronization movement over a sequence of consecutive tracks is possible by using a Planar TrackTrail [▶ 120] object. In such a case, a transition from one Planar track to the next is possible during the synchronization phase or when synchronicity already exists. The deceleration of the Planar mover analogous to the above example with only one Planar track would only occur at the end of the last Planar track, if the movement of the master axis would require it to be exceeded.

6.2.9 Example: "Synchronizing a Planar mover on a track with another Planar mover"

Guided by these instructions you will create a TwinCAT project in which a Planar mover located on a Planar track is coupled to another Planar mover on a parallel Planar track and then follows its setpoints.

Coupling a Planar mover to another Planar mover is largely analogous to coupling a Planar mover to an axis; see Example "Synchronizing a Planar mover on a track with one axis" [▸ 65]. This example is short and builds on the above example.

Creating a Planar mover

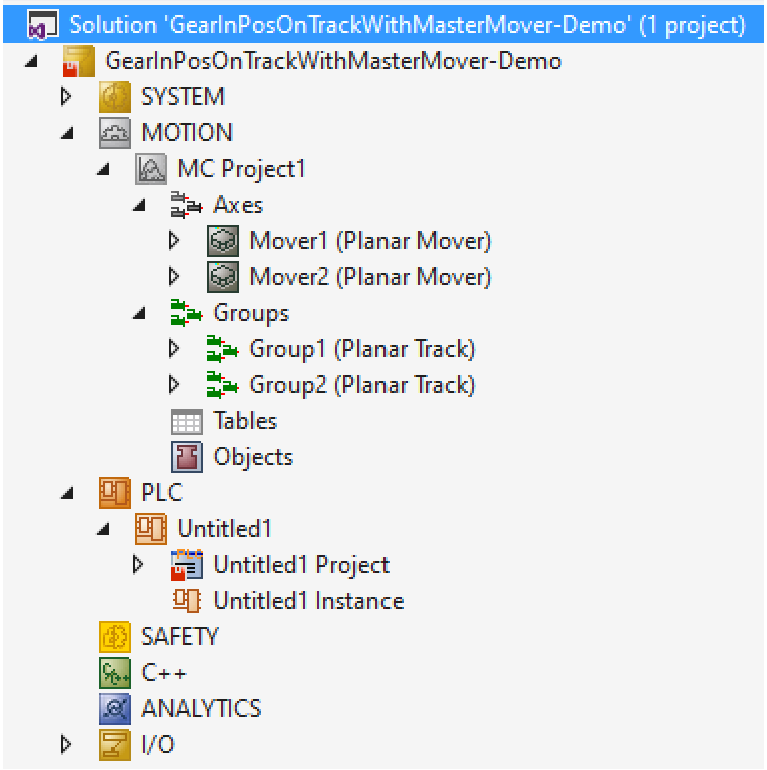
✓ See Configuration [▸ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a Planar track

3. Add two Planar tracks via **Groups > Add New Item…**, see Configuration [▸ 42].

⇨ The Solution Explorer has the following entries:

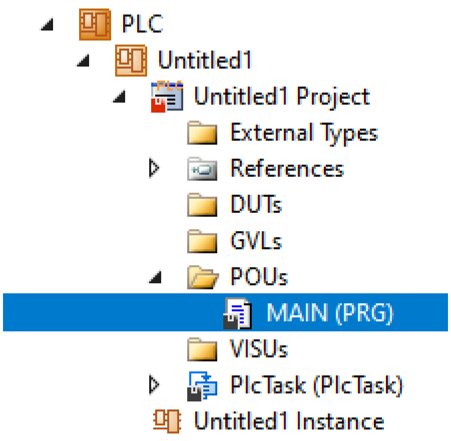


```
Solution 'GearInPosOnTrackWithMasterMover-Demo' (1 project)
◢ GearInPosOnTrackWithMasterMover-Demo
    ▷ SYSTEM
    ◢ MOTION
        ◢ MC Project1
            ◢ Axes
                ▷ Mover1 (Planar Mover)
                ▷ Mover2 (Planar Mover)
            ◢ Groups
                ▷ Group1 (Planar Track)
                ▷ Group2 (Planar Track)
              Tables
              Objects
    ◢ PLC
        ◢ Untitled1
            ▷ Untitled1 Project
              Untitled1 Instance
      SAFETY
      C++
      ANALYTICS
    ▷ I/O
```

Creating a PLC

✓ See preliminary steps under Creating a PLC [▸ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**

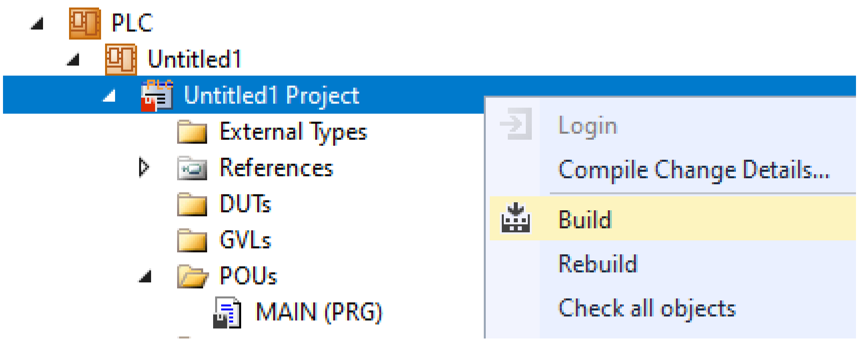


⇨ These represent movers and tracks in the MC Configuration.

2. Create the following variables.

```
PROGRAM MAIN
VAR
    master_mover : MC_PlanarMover;
    slave_mover  : MC_PlanarMover;
    master_track : MC_PlanarTrack;
    slave_track  : MC_PlanarTrack;
    state        : UDINT;
    pos1, pos2   : PositionXYC;
END_VAR
```

3. Build the PLC to create symbols of the "PLC movers" and "PLC tracks".



4. Link the Planar movers and the Planar tracks (see Example "Joining and moving a Planar mover on the track" [▶ 50]).

Programming state machines

With the following state machine, which is programmed in MAIN, the Planar tracks are geometrically defined and activated (states 0 to 3), the Planar movers are activated and coupled to the respective Planar track (states 4 to 11), and the Planar mover acting as master moves on its track (state 12).

Finally, the command to start synchronization with the Master Planar Mover (GearInPosOnTrackWithMasterMover [▶ 148]) is sent to the Slave Planar Mover (state 13). After the END_CASE statement, the Planar objects are updated cyclically.

```
CASE state OF
  0:
    pos1.SetValuesXYC(100, 620, 0);
    pos2.SetValuesXYC(860, 620, 0);
    master_track.AppendLine(0, pos1, pos2);
    master_track.Enable(0);
    state := state + 1;
  1:
    IF master_track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  2:
    pos1.SetValuesXYC(100, 100, 0);
    pos2.SetValuesXYC(860, 100, 0);
    slave_track.AppendLine(0, pos1, pos2);
    slave_track.Enable(0);
    state := state + 1;
  3:
    IF slave_track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  4:
    master_mover.Enable(0);
    state := state + 1;
  5:
    IF master_mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  6:
    master_mover.JoinTrack(0, master_track, 0, 0);
    state := state + 1;
  7:
    IF master_mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
    END_IF
  8:
    slave_mover.Enable(0);
    state := state + 1;
  9:
    IF slave_mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  10:
    slave_mover.JoinTrack(0, slave_track, 0, 0);
    state := state + 1;
  11:
    IF slave_mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
    END_IF
  12:
    master_mover.MoveOnTrack(0, 0, 500.0, 0, 0);
    state := state + 1;
  13:
    slave_mover.GearInPosOnTrackWithMasterMover(0, master_mover, 0, 100.0, master_track, 100.0, slav
e_track, 0, 0);
    state := state + 1;
END_CASE

master_mover.Update();
slave_mover.Update();
master_track.Update();
slave_track.Update();
```

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

The Master Planar Mover will move to the given target position (in this case 500) on the specified Planar track, and the Slave Planar Mover will follow its movement. The positions of the Planar movers can be tracked in the online view (by clicking the button).

The Slave Planar Mover stops at position 500, since the Master Planar Mover also reaches zero dynamics here.

In the function call in State 13, the sync positions of the master (arguments 4 and 5) or slave (arguments 6 and 7) are passed to the Slave Planar Mover. These are the respective positions at which the slave becomes synchronous with the master, i.e. at which it reaches its dynamic values. In fact, here as well as in Example "Synchronizing a Planar mover on a track with one axis" [▶ 65], it is possible for the slave to get in sync with its master significantly sooner. Like with synchronization with an axis, a special Specialized feedback types [▶ 119] object is required for monitoring synchronicity status and possible errors.

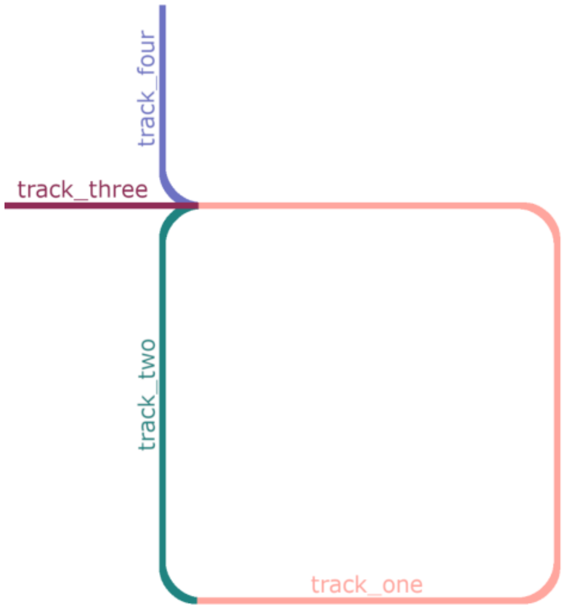
Like with synchronization with a master axis, the synchronization movement of the slave can be programmed over several tracks by specifying a Planar TrackTrail [▶ 120] object.

If the Master Planar Mover moves across a track boundary during an active synchronization command, the position it passes to its slave is simply summed across the track boundary.

If a master sync position is to be specified on a Planar track passed by the Master Planar Mover in the future, make sure that the Master Planar Mover has already commanded a move involving that Planar track at the time the GearInPosOnTrackWithMasterMover [▶ 148] command is sent.

6.2.10 Example "Connecting Planar tracks to a network"

Using this guide, you will be able to create a TwinCAT project that connects four Planar tracks to a network.



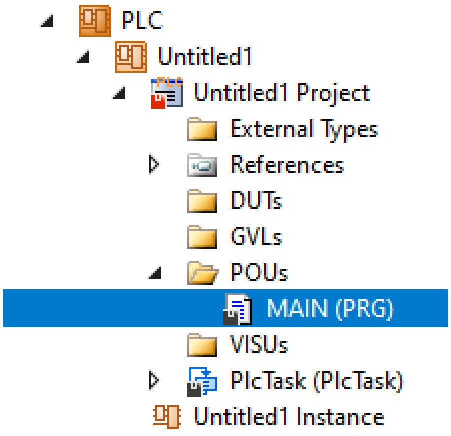
Creating a Planar track

1. Add four Planar tracks via **Groups > Add New Item…**, see Configuration [▶ 42].

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



⇨ These represent movers and tracks in the MC Configuration.

2. Create four tracks as shown below, plus a state variable for a state machine and two auxiliary positions for the tracks.

```
PROGRAM MAIN
VAR
    track_one, track_two, track_three, track_four : MC_PlanarTrack;
    state : UDINT;
    pos1, pos2 : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.
   ⇨ This program code creates and activates four tracks that are connected to a network, as shown in the illustration above. The so-called "blendings", i.e. the non-linear parts of the track in this example, are generated automatically here. You only specify the straight sections.

```
CASE state OF
  0:
    pos1.SetValuesXY(250, 120);
    pos2.SetValuesXY(650, 120);
    track_one.AppendLine(0, pos1, pos2);
    pos1.SetValuesXY(700, 170);
    pos2.SetValuesXY(800, 450);
    track_one.AppendLine(0, pos1, pos2);
    pos1.SetValuesXY(650, 500);
    pos2.SetValuesXY(250, 500);
    track_one.AppendLine(0, pos1, pos2);
    state := 1;
  1:
    pos1.SetValuesXY(200, 450);
    pos2.SetValuesXY(200, 170);
    track_two.StartFromTrack(0,track_one);
    track_two.AppendLine(0, pos1, pos2);
    track_two.EndAtTrack(0,track_one);
    state := 2;
  2:
    pos1.SetValuesXY(200, 500);
    pos2.SetValuesXY(120, 500);
    track_three.StartFromTrack(0,track_one);
    track_three.AppendLine(0, pos1, pos2);
    state := 3;
  3:
    pos1.SetValuesXY(200, 550);
    pos2.SetValuesXY(200, 750);
    track_four.StartFromTrack(0,track_one);
    track_four.AppendLine(0, pos1, pos2);
    state := 4;
  4:
    track_one.Enable(0);
    track_two.Enable(0);
    track_three.Enable(0);
    track_four.Enable(0);
    state := 5;
  5:
    IF track_one.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled AND
    track_two.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled AND
    track_three.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled AND
    track_four.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 6;
    END_IF
END_CASE
```

Tracks must be C^2-continuous at all points. This means that their positions, directions, and curvatures must merge seamlessly. The automatically generated blendings take this requirement into account. Even if the corner pieces look like quarter circles, they are not, because circles have a positive (constant) curvature at each point and straight lines have a zero curvature.

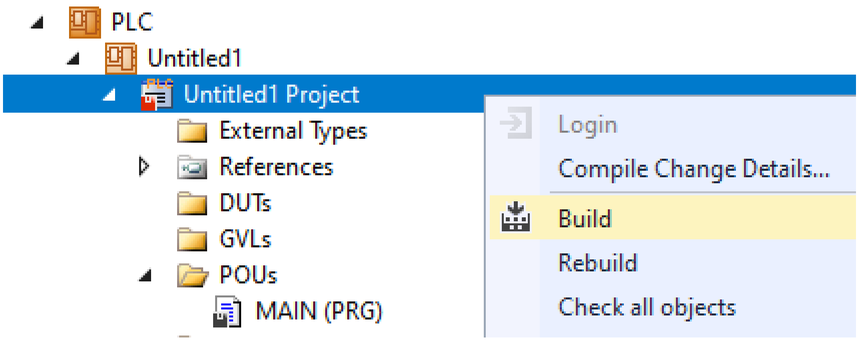
Sending the command

4. To send the command, you must trigger the tracks cyclically with their update method after the END_CASE:

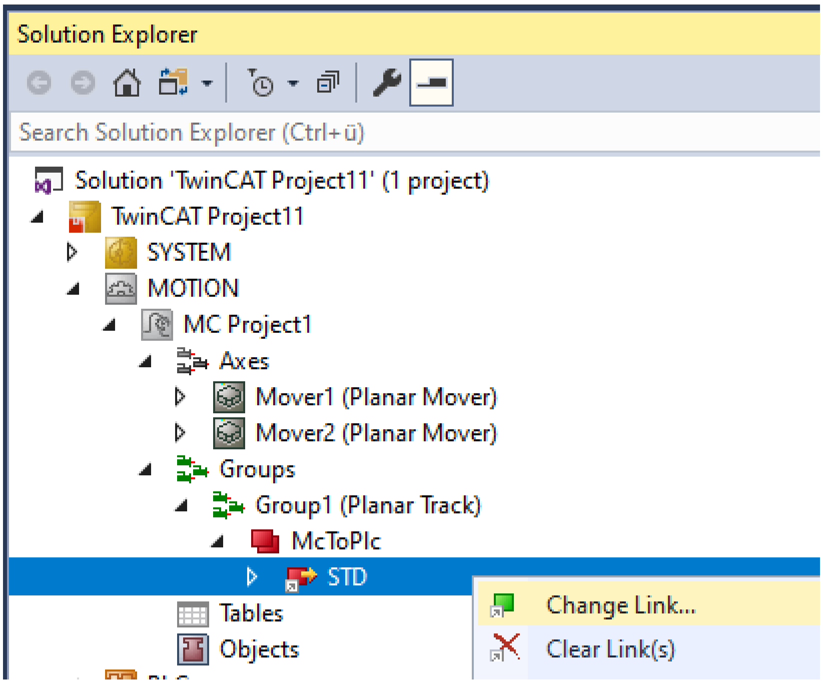
```
track_one.Update();
track_two.Update();
track_three.Update();
track_four.Update();
```

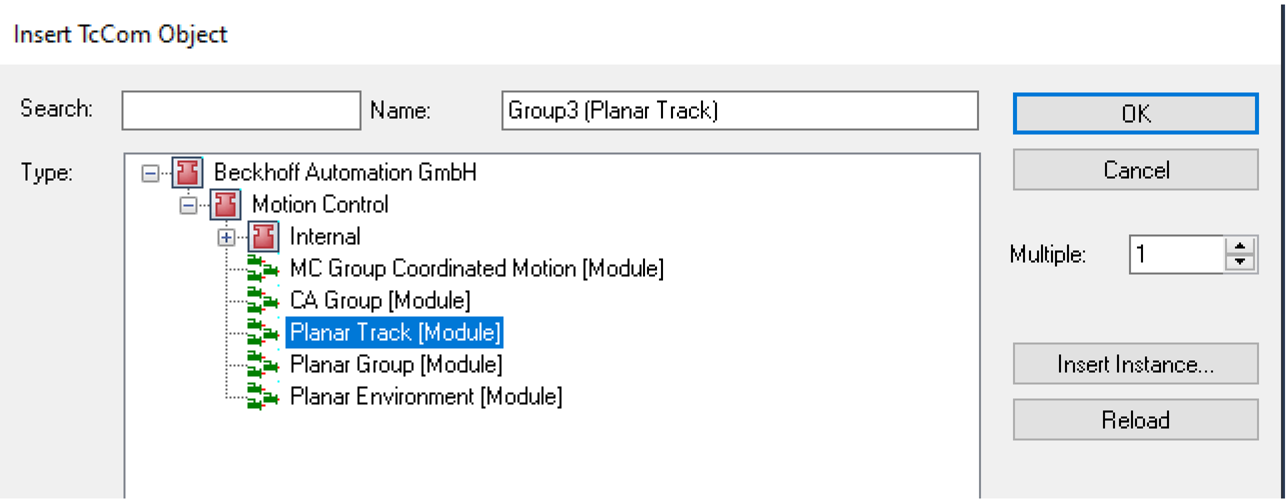
Building the PLC creates symbols of the "PLC mover" and “track”, which can then be linked to the mover and track instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ The tracks must each be linked separately via the following dialog boxes.





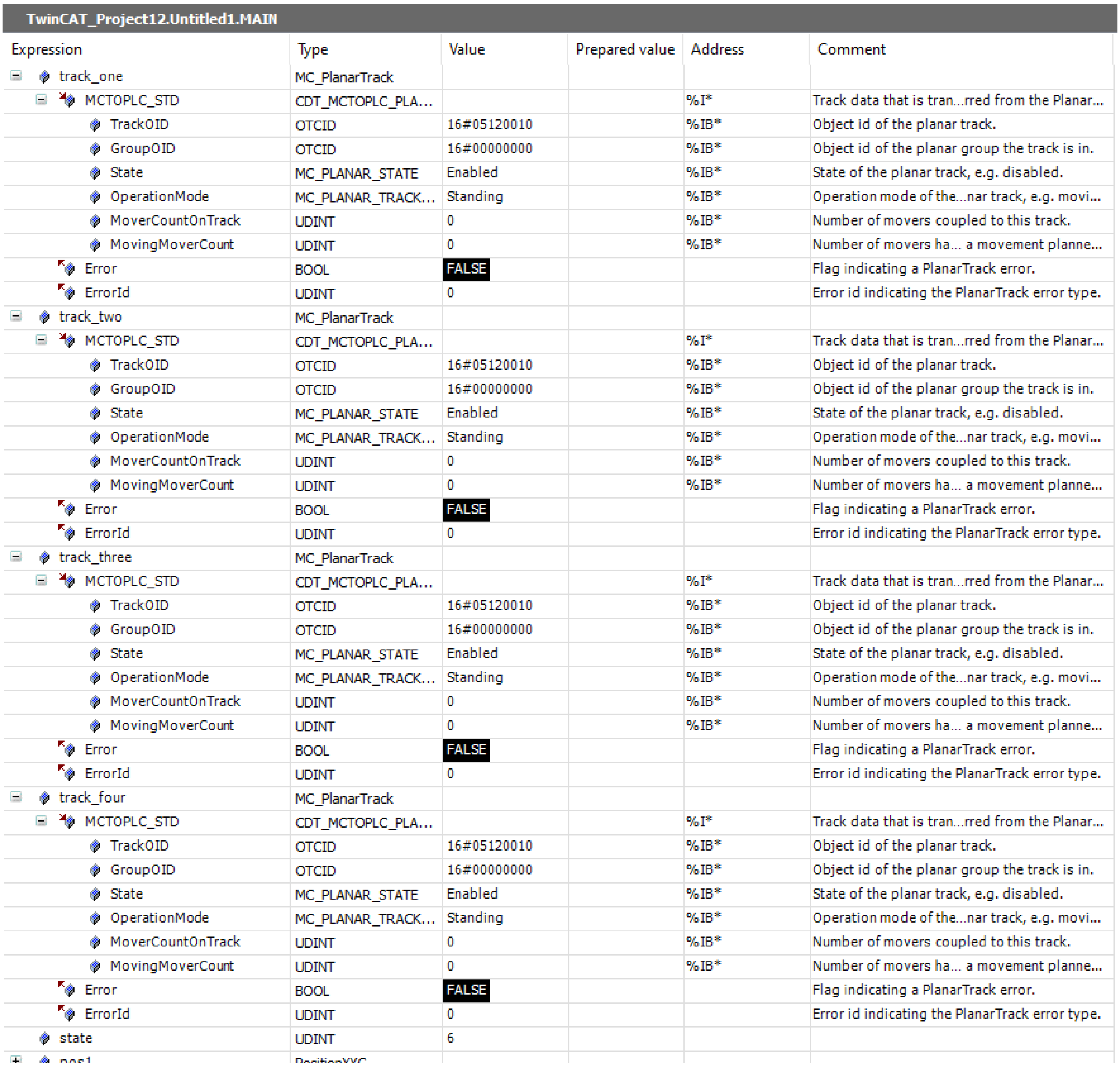
Name: Group3 (Planar Track)

Multiple: 1

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

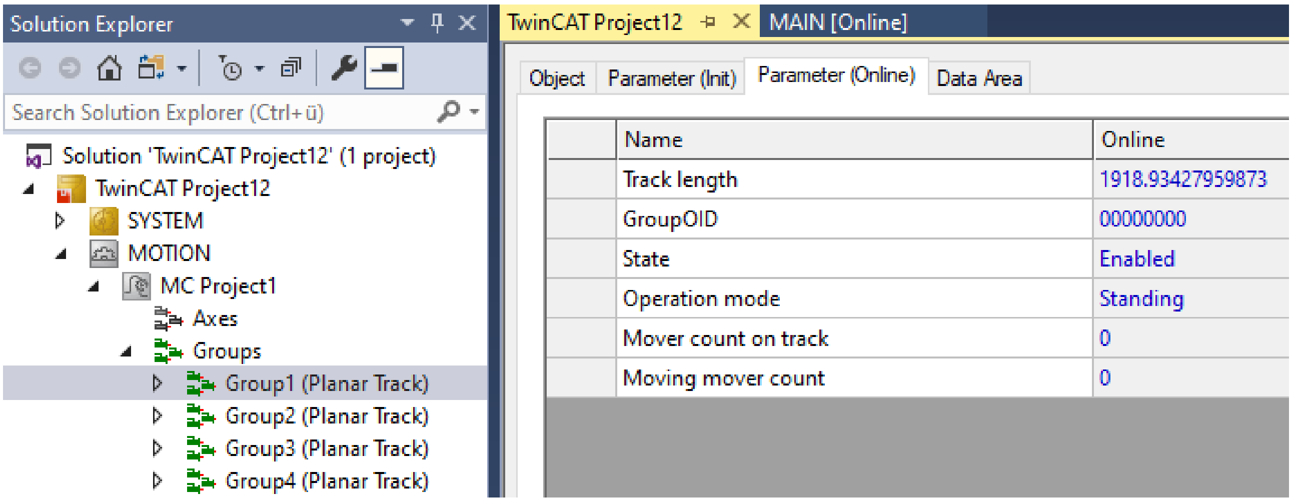
The creation of the track network is finished at the end of the state machine (state = 6).



TwinCAT_Project12.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
track_one	MC_PlanarTrack				
MCTOPLC_STD	CDT_MCTOPLC_PLA...			%I*	Track data that is tran...rred from the Planar...
TrackOID	OTCID	16#05120010		%IB*	Object id of the planar track.
GroupOID	OTCID	16#00000000		%IB*	Object id of the planar group the track is in.
State	MC_PLANAR_STATE	Enabled		%IB*	State of the planar track, e.g. disabled.
OperationMode	MC_PLANAR_TRACK...	Standing		%IB*	Operation mode of the...nar track, e.g. movi...
MoverCountOnTrack	UDINT	0		%IB*	Number of movers coupled to this track.
MovingMoverCount	UDINT	0		%IB*	Number of movers ha... a movement planne...
Error	BOOL	FALSE			Flag indicating a PlanarTrack error.
ErrorId	UDINT	0			Error id indicating the PlanarTrack error type.
track_two	MC_PlanarTrack				
MCTOPLC_STD	CDT_MCTOPLC_PLA...			%I*	Track data that is tran...rred from the Planar...
TrackOID	OTCID	16#05120010		%IB*	Object id of the planar track.
GroupOID	OTCID	16#00000000		%IB*	Object id of the planar group the track is in.
State	MC_PLANAR_STATE	Enabled		%IB*	State of the planar track, e.g. disabled.
OperationMode	MC_PLANAR_TRACK...	Standing		%IB*	Operation mode of the...nar track, e.g. movi...
MoverCountOnTrack	UDINT	0		%IB*	Number of movers coupled to this track.
MovingMoverCount	UDINT	0		%IB*	Number of movers ha... a movement planne...
Error	BOOL	FALSE			Flag indicating a PlanarTrack error.
ErrorId	UDINT	0			Error id indicating the PlanarTrack error type.
track_three	MC_PlanarTrack				
MCTOPLC_STD	CDT_MCTOPLC_PLA...			%I*	Track data that is tran...rred from the Planar...
TrackOID	OTCID	16#05120010		%IB*	Object id of the planar track.
GroupOID	OTCID	16#00000000		%IB*	Object id of the planar group the track is in.
State	MC_PLANAR_STATE	Enabled		%IB*	State of the planar track, e.g. disabled.
OperationMode	MC_PLANAR_TRACK...	Standing		%IB*	Operation mode of the...nar track, e.g. movi...
MoverCountOnTrack	UDINT	0		%IB*	Number of movers coupled to this track.
MovingMoverCount	UDINT	0		%IB*	Number of movers ha... a movement planne...
Error	BOOL	FALSE			Flag indicating a PlanarTrack error.
ErrorId	UDINT	0			Error id indicating the PlanarTrack error type.
track_four	MC_PlanarTrack				
MCTOPLC_STD	CDT_MCTOPLC_PLA...			%I*	Track data that is tran...rred from the Planar...
TrackOID	OTCID	16#05120010		%IB*	Object id of the planar track.
GroupOID	OTCID	16#00000000		%IB*	Object id of the planar group the track is in.
State	MC_PLANAR_STATE	Enabled		%IB*	State of the planar track, e.g. disabled.
OperationMode	MC_PLANAR_TRACK...	Standing		%IB*	Operation mode of the...nar track, e.g. movi...
MoverCountOnTrack	UDINT	0		%IB*	Number of movers coupled to this track.
MovingMoverCount	UDINT	0		%IB*	Number of movers ha... a movement planne...
Error	BOOL	FALSE			Flag indicating a PlanarTrack error.
ErrorId	UDINT	0			Error id indicating the PlanarTrack error type.
state	UDINT	6			
pos1	PositionXYC				

The length of each track is in the online parameters of the TCom objects in the MC Project.



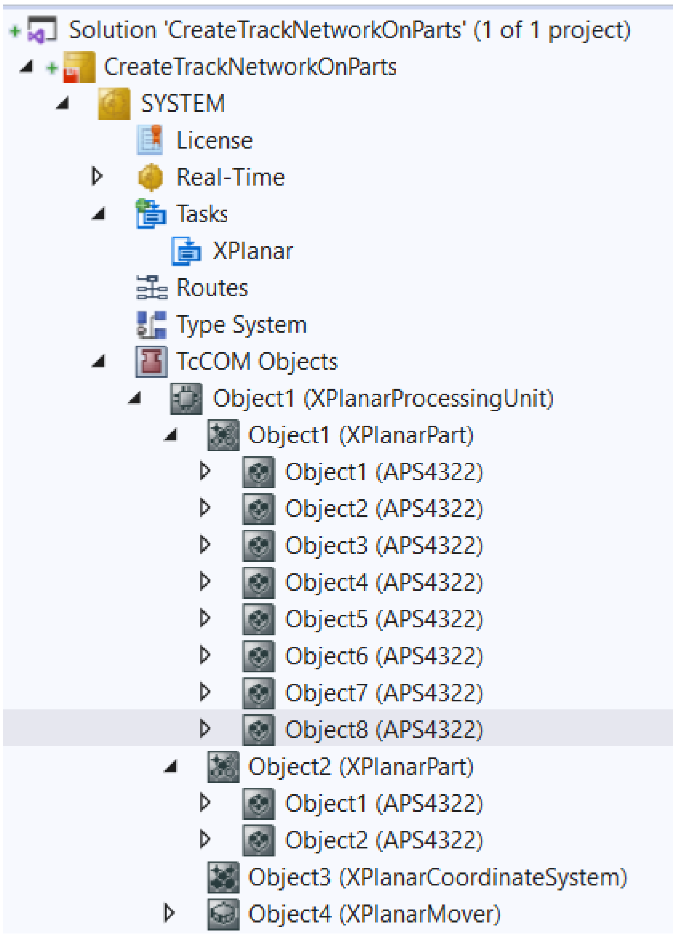
Name	Online
Track length	1918.93427959873
GroupOID	00000000
State	Enabled
Operation mode	Standing
Mover count on track	0
Moving mover count	0

6.2.11 Example "Connecting Planar tracks to network on Planar parts"

In this example, a network of Planar tracks is created on moving Planar parts.

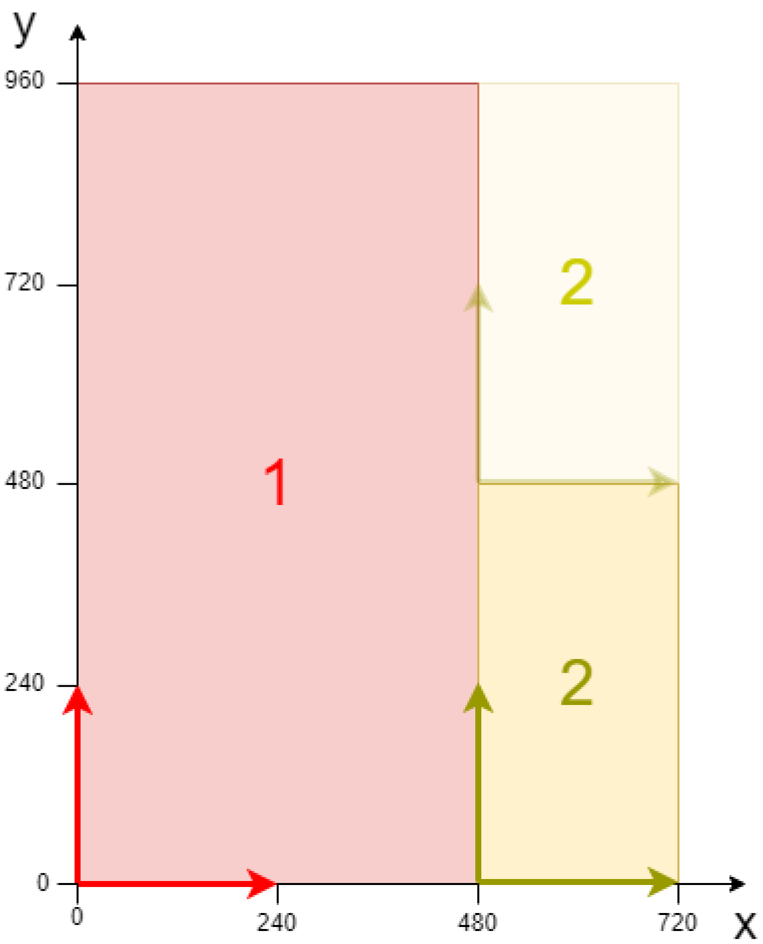
Starting point

We start with a solution that contains a fully configured XPlanar Processing Unit. Two parts, a coordinate system and a mover are created under the XPlanar Processing Unit. There are 8 tiles under the first part and 2 tiles under the second part.



```
Solution 'CreateTrackNetworkOnParts' (1 of 1 project)
  CreateTrackNetworkOnParts
    SYSTEM
      License
      Real-Time
      Tasks
        XPlanar
      Routes
      Type System
      TcCOM Objects
        Object1 (XPlanarProcessingUnit)
          Object1 (XPlanarPart)
            Object1 (APS4322)
            Object2 (APS4322)
            Object3 (APS4322)
            Object4 (APS4322)
            Object5 (APS4322)
            Object6 (APS4322)
            Object7 (APS4322)
            Object8 (APS4322)
          Object2 (XPlanarPart)
            Object1 (APS4322)
            Object2 (APS4322)
          Object3 (XPlanarCoordinateSystem)
          Object4 (XPlanarMover)
```

The following geometric situation is set: the two parts are next to each other and part 2 can occupy two positions. This means that there are two different connections between the parts, depending on the part positions.



The example is developed on the basis of this configuration. The creation of the initial situation is described in the XPlanar Processing Unit documentation.

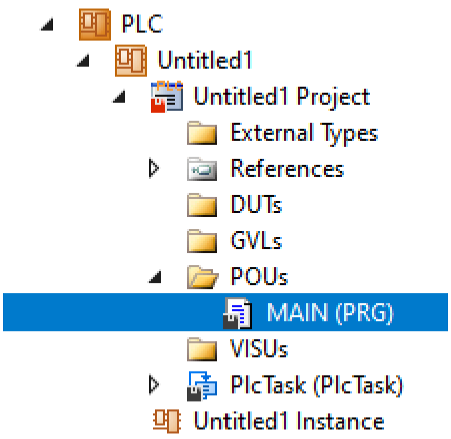
Creating Planar tracks and a Planar environment

1. Create a Planar environment, see Configuration [▶ 96].
2. Set the initial parameter XPlanar processing unit OID to the object ID of the XPlanar Processing Unit. This activates the Part feature for all **MC Configuration** objects (especially for the created Planar mover).
3. Add three Planar tracks via **Groups > Add New Item…**, see Configuration [▶ 42].
4. Set the initial parameter "PartOID" of the three tracks to the corresponding parts. In this example, the first and second tracks are on part 1 and the third track is on part 2.

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



⇨ These represent movers and tracks in the MC Configuration.

2. Create three planar tracks as shown below, a state variable for a state machine and two auxiliary positions for the tracks.

```
PROGRAM MAIN
VAR
    track_one, track_two, track_three : MC_PlanarTrack;
    state : UDINT;
    pos1, pos2 : PositionXYC;
    part_one_oid : OTCID := 16#01010020;
    part_two_oid : OTCID := 16#01010030;
    start_options : ST_StartFromTrackAdvancedOptions;
    end_options : ST_EndAtTrackAdvancedOptions;
END_VAR
```

3. Then program a sequence in MAIN.

   ⇨ This program code creates three tracks on the parts. Depending on the active part position, the tracks are connected at different positions. First, track 3 is created on part 2 (state=0). Track 2 is then started at track 3, with part 2 in the upper position (index=2). Track 2 also ends at track 3, but part 2 is in the lower position (index=1). Finally, track 1 is started at track 3, part 2 is in the lower position (index=1), and finished, part 2 is in the upper position (index=2).

```
CASE state OF
  0:
    pos1.SetValuesXYCReferenceId(0, 360, 0, part_two_oid);
    pos2.SetValuesXYCReferenceId(80, 360, 0, part_two_oid);
    track_three.AppendLine(0, pos1, pos2);
    pos1.SetValuesXYCReferenceId(120, 320, 0, part_two_oid);
    pos2.SetValuesXYCReferenceId(120, 160, 0, part_two_oid);
    track_three.AppendLine(0, pos1, pos2);
    pos1.SetValuesXYCReferenceId(80, 120, 0, part_two_oid);
    pos2.SetValuesXYCReferenceId(0, 120, 0, part_two_oid);
    track_three.AppendLine(0, pos1, pos2);
    state := 1;
  1:
    start_options.thisTrackPartPositionIndex := 1;
    start_options.otherTrackPartPositionIndex := 2;
    track_two.StartFromTrackAdvanced(0, track_three, start_options);
    pos1.SetValuesXYCReferenceId(440, 600, 0, part_one_oid);
    pos2.SetValuesXYCReferenceId(400, 600, 0, part_one_oid);
    track_two.AppendLine(0, pos1, pos2);
    pos1.SetValuesXYCReferenceId(360, 560, 0, part_one_oid);
    pos2.SetValuesXYCReferenceId(360, 400, 0, part_one_oid);
    track_two.AppendLine(0, pos1, pos2);
    pos1.SetValuesXYCReferenceId(400, 360, 0, part_one_oid);
    pos2.SetValuesXYCReferenceId(440, 360, 0, part_one_oid);
    track_two.AppendLine(0, pos1, pos2);
    end_options.thisTrackPartPositionIndex := 1;
    end_options.otherTrackPartPositionIndex := 1;
    track_two.EndAtTrackAdvanced(0, track_three, end_options);
    state := 2;
  2:
    start_options.thisTrackPartPositionIndex := 1;
    start_options.otherTrackPartPositionIndex := 1;
    track_one.StartFromTrackAdvanced(0, track_three, start_options);
    pos1.SetValuesXYCReferenceId(440, 120, 0, part_one_oid);
    pos2.SetValuesXYCReferenceId(160, 120, 0, part_one_oid);
    track_one.AppendLine(0, pos1, pos2);
    pos1.SetValuesXYCReferenceId(120, 160, 0, part_one_oid);
    pos2.SetValuesXYCReferenceId(120, 800, 0, part_one_oid);
    track_one.AppendLine(0, pos1, pos2);
    pos1.SetValuesXYCReferenceId(160, 840, 0, part_one_oid);
    pos2.SetValuesXYCReferenceId(440, 840, 0, part_one_oid);
    track_one.AppendLine(0, pos1, pos2);
    end_options.thisTrackPartPositionIndex := 1;
    end_options.otherTrackPartPositionIndex := 2;
    track_one.EndAtTrackAdvanced(0, track_three, end_options);
    state := 3;
END_CASE
```

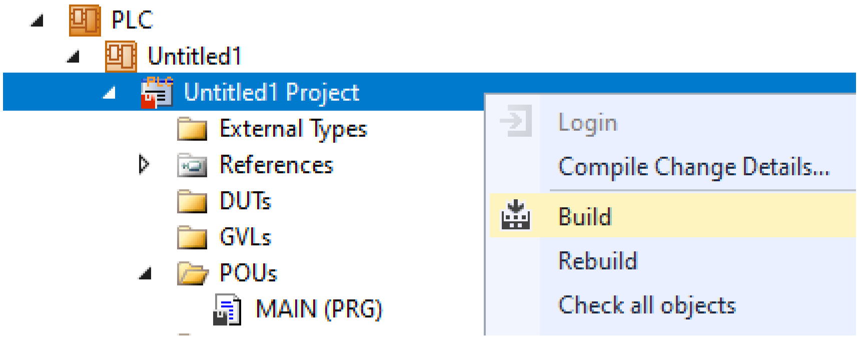
Sending the command

4. To send the commands, you must call the track cyclically with its update method after the END_CASE:

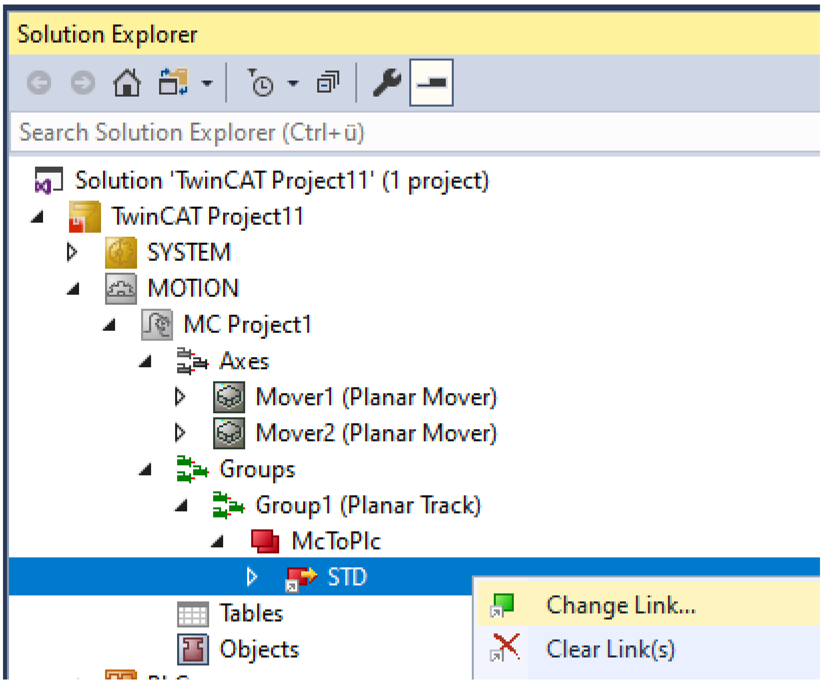
```
track_one.Update();
track_two.Update();
track_three.Update();
```

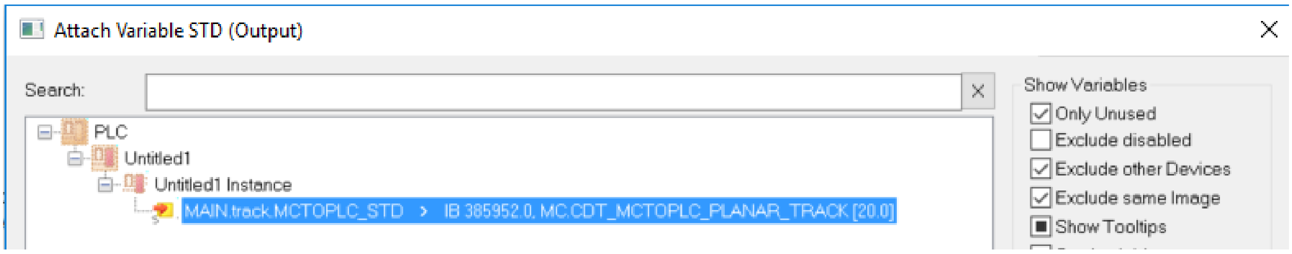
Building the PLC creates symbols of the "PLC mover" and “track”, which can then be linked to the mover and track instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ The tracks can now be linked via the following dialog boxes.





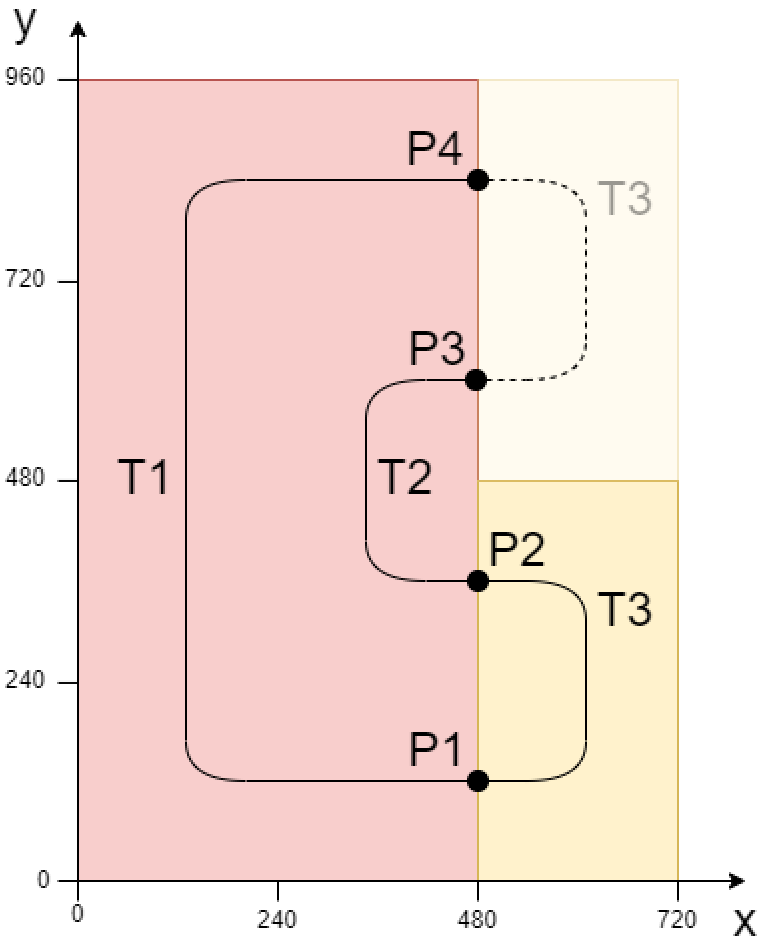
MAIN.track.MCTOPLC_STD > IB 385952.0, MC.CDT_MCTOPLC_PLANAR_TRACK [20.0]

☑ Only Unused
☐ Exclude disabled
☑ Exclude other Devices
☑ Exclude same Image
■ Show Tooltips

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button .
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=3), the three tracks are configured as shown. The start of track 3 is connected to the end of track 2 at position 2 and the end of track 3 is connected to the start of track 1 and position 2 when part 2 is in the lower position. In the upper position, the start of track 3 is connected to the end of track 1 at position 4 and the end of track 3 is connected to the start of track 2 and position 1. Track 2 and track 3 are the same length.



6.2.12 Example: "Following a Planar mover through a Track Network"

Guided by these instructions, you will create a TwinCAT project in which a Planar mover located on a Planar track follows a preceding Planar mover on the same Planar track on its path through a track network.

Following through a track network is realized by the command GearInPosOnTrackWithMasterMover [▶ 148], which is described in more detail in Example: "Synchronizing a Planar mover on a track with another Planar mover" [▶ 72]. Creating and building a network of Planar tracks is explained in more detail in the Example "Connecting Planar tracks to a network" [▶ 75]. This example is short and builds on the above examples.

Creating a Planar mover

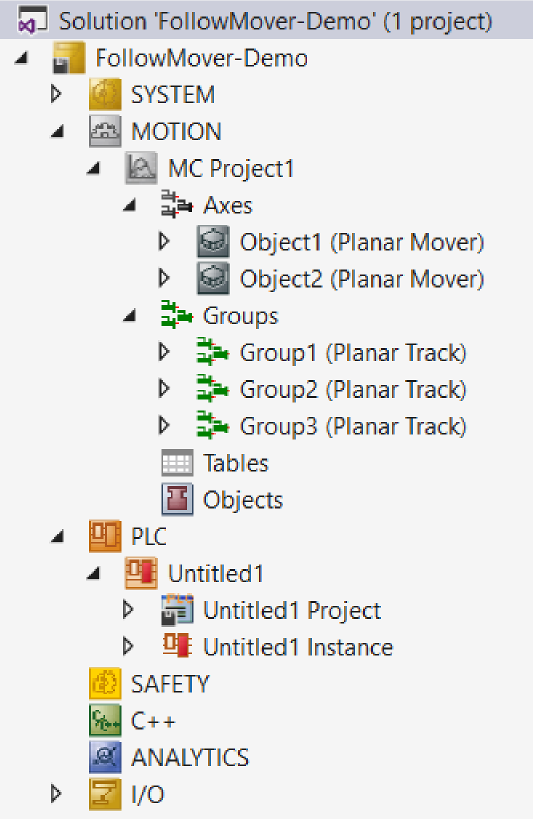
✓ See Configuration [▶ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a Planar track

3. Add three Planar tracks via **Groups > Add New Item…**, see Configuration [▶ 42].

⇨ The Solution Explorer has the following entries:

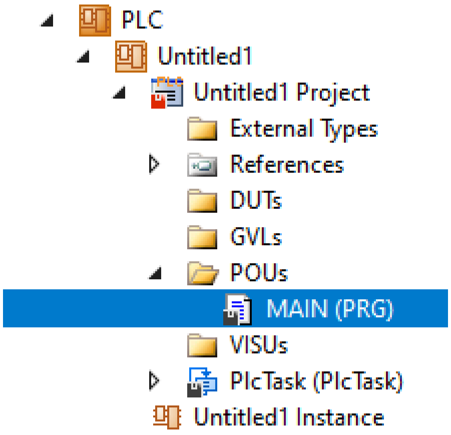


Solution 'FollowMover-Demo' (1 project)
- FollowMover-Demo
  - SYSTEM
  - MOTION
    - MC Project1
      - Axes
        - Object1 (Planar Mover)
        - Object2 (Planar Mover)
      - Groups
        - Group1 (Planar Track)
        - Group2 (Planar Track)
        - Group3 (Planar Track)
      - Tables
      - Objects
  - PLC
    - Untitled1
      - Untitled1 Project
      - Untitled1 Instance
  - SAFETY
  - C++
  - ANALYTICS
  - I/O

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via **MAIN.**



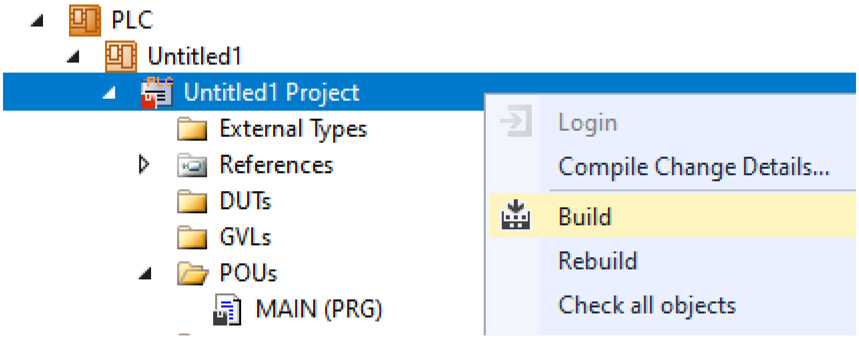
- PLC
  - Untitled1
    - Untitled1 Project
      - External Types
      - References
      - DUTs
      - GVLs
      - POUs
        - MAIN (PRG)
      - VISUs
      - PlcTask (PlcTask)
    - Untitled1 Instance

⇨ These represent movers and tracks in the MC Configuration.

2. Create the following variables.

```
PROGRAM MAIN
VAR
    master_mover   : MC_PlanarMover;
    slave_mover    : MC_PlanarMover;
    track_in       : MC_PlanarTrack;
    track_out1     : MC_PlanarTrack;
    track_out2     : MC_PlanarTrack;
    move_feedback  : MC_PlanarFeedback;
    options        : ST_GearInPosOnTrackWithMasterMoverOptions;
    state          : UDINT;
    pos1, pos2     : PositionXYC;
END_VAR
```

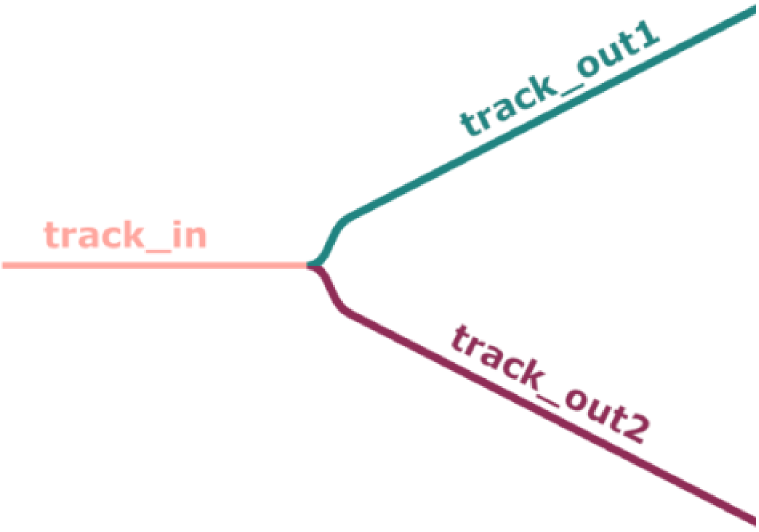
3. Build the PLC to create symbols of the "PLC movers" and "PLC tracks".



4. Link the Planar movers and the Planar tracks (see Example "Joining and moving a Planar mover on the track" [▶ 50]).

Programming state machines

With the following state machine, which is programmed in MAIN, first the Planar tracks are geometrically defined and activated (states 0 to 7), so that they represent the following switch configuration:



In states 8 to 19 the two Planar movers are activated, coupled to the Planar track in front of the switch (`track_in`) and moved to position 200 (`master_mover`) or 0 (`slave_mover`). The Master Planar Mover is then commanded to position 500 on the upper of the two branching Planar tracks (`track_out1`) (State 20). Finally, in State 21, the GearInPosOnTrackWithMasterMover [▶ 148] command is sent to the Slave Planar Mover. As usual, the Planar objects are updated cyclically after the `END_CASE` statement.

```
CASE state OF
  0:
    pos1.SetValuesXYC(100, 360, 0);
    pos2.SetValuesXYC(400, 360, 0);
    track_in.AppendLine(0, pos1, pos2);
    track_in.Enable(0);
    state := state + 1;
  1:
    IF track_in.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  2:
    track_out1.StartFromTrack(0, track_in);
    state := state + 1;
  3:
    pos1.SetValuesXYC(450, 410, 0);
    pos2.SetValuesXYC(860, 620, 0);
    track_out1.AppendLine(0, pos1, pos2);
    track_out1.Enable(0);
    state := state + 1;
  4:
    IF track_out1.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  5:
    track_out2.StartFromTrack(0, track_in);
    state := state + 1;
  6:
    pos1.SetValuesXYC(450, 310, 0);
    pos2.SetValuesXYC(860, 100, 0);
    track_out2.AppendLine(0, pos1, pos2);
    track_out2.Enable(0);
    state := state + 1;
  7:
    IF track_out2.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  8:
    master_mover.Enable(0);
    state := state + 1;
  9:
    IF master_mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  10:
    master_mover.JoinTrack(0, track_in, 0, 0);
    state := state + 1;
  11:
    IF master_mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
    END_IF
  12:
    master_mover.MoveOnTrack(move_feedback, track_in, 200, 0, 0);
    state := state + 1;
  13:
    IF move_feedback.Done THEN
      state := state + 1;
    END_IF
  14:
    slave_mover.Enable(0);
    state := state + 1;
  15:
    IF slave_mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  16:
    slave_mover.JoinTrack(0, track_in, 0, 0);
    state := state + 1;
  17:
    IF slave_mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
    END_IF
  18:
    slave_mover.MoveOnTrack(move_feedback, track_in, 0, 0, 0);
    state := state + 1;
  19:
    IF move_feedback.Done THEN
      state := state + 1;
    END_IF
  20:
    master_mover.MoveOnTrack(0, track_out1, 500, 0, 0);
    state := state + 1;
  21:
    options.followMover := TRUE;
    slave_mover.GearInPosOnTrackWithMasterMover(0, master_mover, 0, 210, track_in, 10, track_in, 0, 
options);
    state := state + 1;
END_CASE

master_mover.Update();
slave_mover.Update();
```

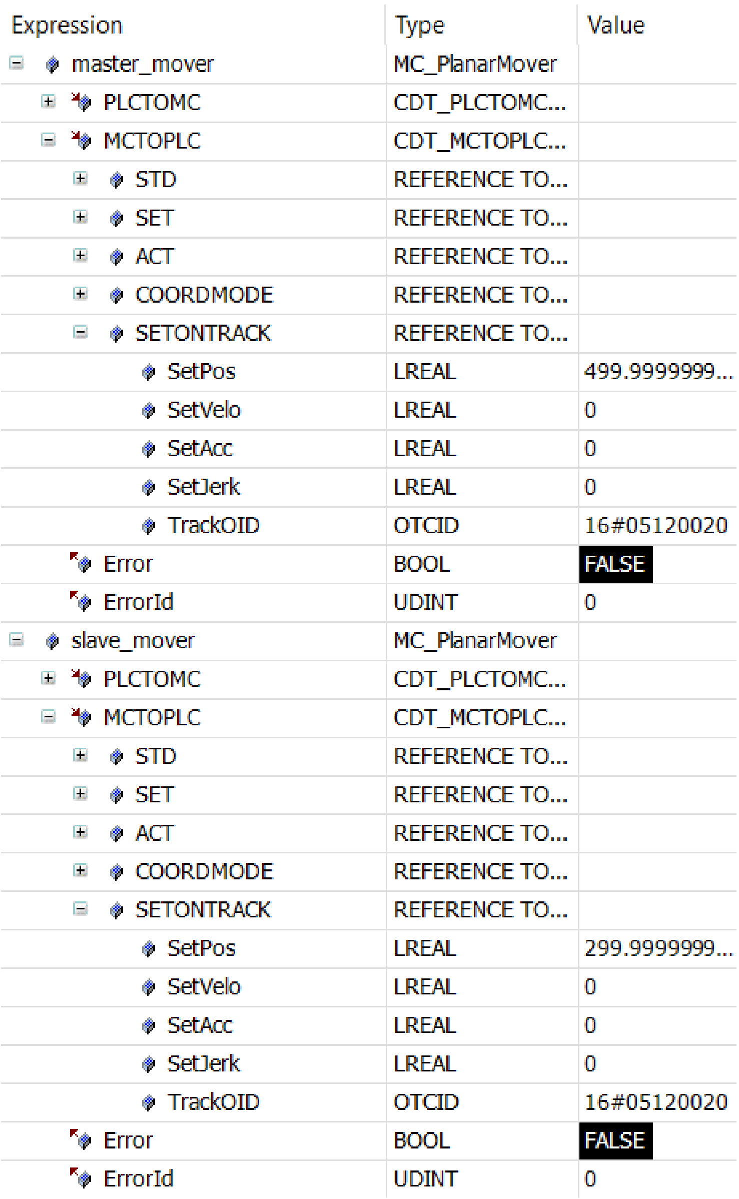
```
track_in.Update();
track_out1.Update();
track_out2.Update();
move_feedback.Update();
```

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

The Master Planar Mover will move to the given target position (in this case 500) on the specified Planar track, and the Slave Planar Mover will follow its movement. The positions of the Planar movers can be tracked in the online view (by clicking the button).

Since the positions 210 for the master and 10 for the slave were specified as the sync positions of the two Planar movers in the function call in State 21, the Slave Planar Mover will follow its master through the network at a distance of 200. It stops at position 300 on the upper of the two branching Planar tracks (on which the Master Planar Mover is also located), which can be checked in the online view:



Expression	Type	Value
⊟ master_mover	MC_PlanarMover	
⊞ PLCTOMC	CDT_PLCTOMC...	
⊟ MCTOPLC	CDT_MCTOPLC...	
⊞ STD	REFERENCE TO...	
⊞ SET	REFERENCE TO...	
⊞ ACT	REFERENCE TO...	
⊞ COORDMODE	REFERENCE TO...	
⊟ SETONTRACK	REFERENCE TO...	
SetPos	LREAL	499.9999999...
SetVelo	LREAL	0
SetAcc	LREAL	0
SetJerk	LREAL	0
TrackOID	OTCID	16#05120020
Error	BOOL	FALSE
ErrorId	UDINT	0
⊟ slave_mover	MC_PlanarMover	
⊞ PLCTOMC	CDT_PLCTOMC...	
⊟ MCTOPLC	CDT_MCTOPLC...	
⊞ STD	REFERENCE TO...	
⊞ SET	REFERENCE TO...	
⊞ ACT	REFERENCE TO...	
⊞ COORDMODE	REFERENCE TO...	
⊟ SETONTRACK	REFERENCE TO...	
SetPos	LREAL	299.9999999...
SetVelo	LREAL	0
SetAcc	LREAL	0
SetJerk	LREAL	0
TrackOID	OTCID	16#05120020
Error	BOOL	FALSE
ErrorId	UDINT	0

Note that setting the "FollowMover" option in the Options object and passing it in the function call in State 21 avoids the need to specify a PlanarTrackTrail [▶ 120] object. The path through the network that the Slave Planar Mover should take does not have to be explicitly determined, since it automatically follows the Master Planar Mover and turns to the correct Planar track at the switch. With the set option this behavior is also reproduced in a larger network, where the Master Planar Mover moves across multiple track boundaries.

6.2.13 Options for the "StartFromTrackAdvanced" and "EndAtTrackAdvanced" commands

As described in the introduction and shown in the examples, it is not always clear what a StartFromTrack [▶ 165] or a EndAtTrack [▶ 165] command means. Therefore, both commands are offered in "Advanced" variants (StartFromTrackAdvanced [▶ 166], EndAtTrackAdvanced [▶ 167]) and extended by an option argument. This extension is also mapped in the init parameters.

The option argument has three components: "thisTrackPartPositionIndex", "otherTrackPartPositionIndex" and "linkOnlyInSpecifiedPartPositions". The first component "thisTrackPartPositionIndex" specifies the position of the part on which the called track is located. The unique index of the position must be specified. The second component has the same meaning for the track that is passed as an argument. The third component specifies whether both tracks are only connected in the specified position or also in all other geometrically compatible positions.

If both indices are zero and the flag `false`, the behavior is identical to the previous commands. This can also be achieved by simply passing "0" as an option argument for the "Advanced" commands.

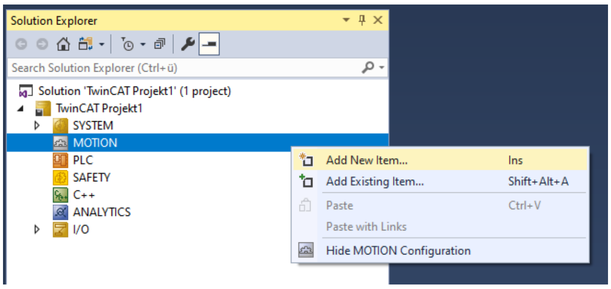
6.3 Planar group

The Planar group is a software object that prevents collisions between Planar movers as well as collisions of Planar movers with the boundary of the stator area on the two-dimensional XPlanar stator area. To do this, the 2D areas of all objects in the group are blocked. When a motion command is transferred to a mover, the required area is requested from the Planar group and the motion command is rejected if this area would collide with already reserved areas. If the motion command can be executed, the area is added to the set of reserved area and blocked accordingly.

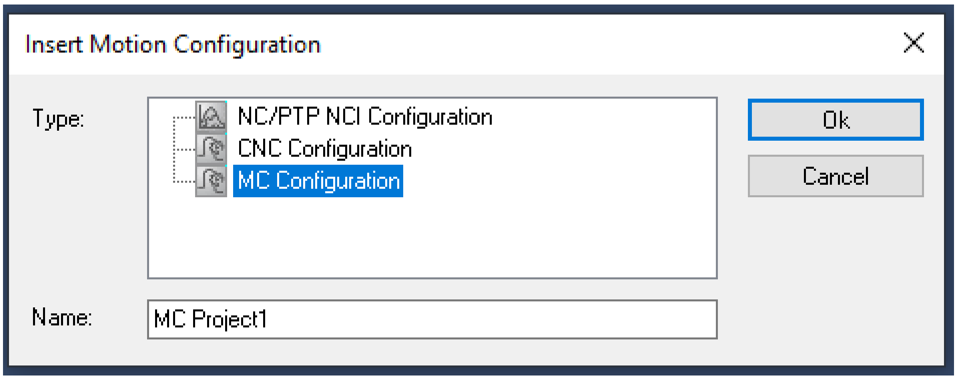
6.3.1 Configuration

✓ In order to create a Planar group, an **MC Configuration** must first be created.

1. Select **MOTION** > **Add New Item…** .


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Projekt1' (1 project)
TwinCAT Projekt1
SYSTEM
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
Add New Item... Ins
Add Existing Item... Shift+Alt+A
Paste Ctrl+V
Paste with Links
Hide MOTION Configuration

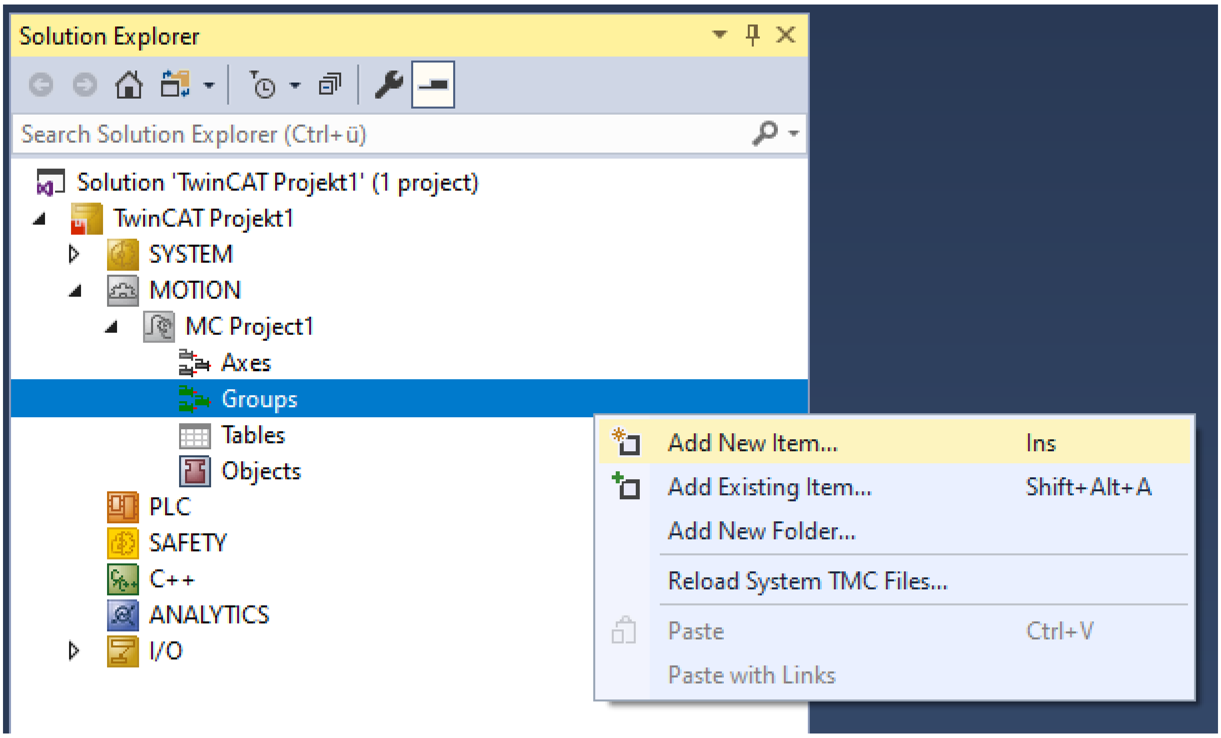
2. In the following dialog box, select **MC Configuration** and confirm with **OK**.



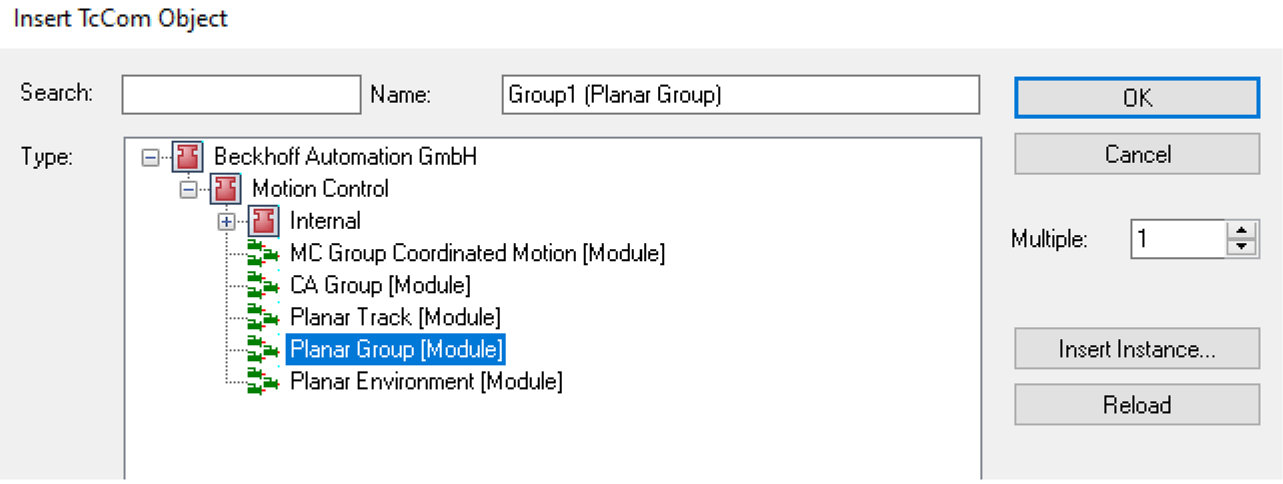
Name: MC Project1

⇨ You have created an MC Project.

3. Select **MC Project** > **Groups** > **Add New Item…**.



4. In the following dialog box, create one (or more) Planar groups and confirm with **OK**.



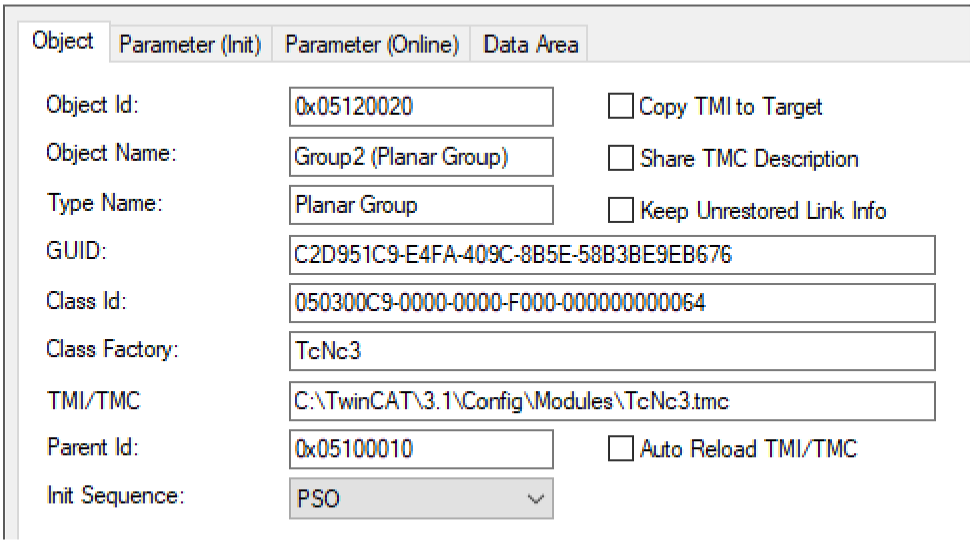
⇨ The Planar group is now created and can be parameterized.

Open detailed description

- Select the Planar group in the tree and double-click it.

Meaning of the individual tabs

Object: General information (name, type, ID and so on) is shown here.



Object Id: 0x05120020

Object Name: Group2 (Planar Group)

Type Name: Planar Group

GUID: C2D951C9-E4FA-409C-8B5E-58B3BE9EB676

Class Id: 050300C9-0000-0000-F000-000000000064

Class Factory: TcNc3

TMI/TMC: C:\TwinCAT\3.1\Config\Modules\TcNc3.tmc

Parent Id: 0x05100010

Init Sequence: PSO

□ Copy TMI to Target

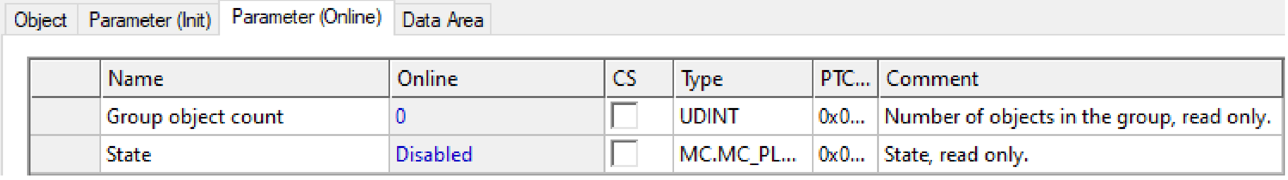
□ Share TMC Description

□ Keep Unrestored Link Info

□ Auto Reload TMI/TMC

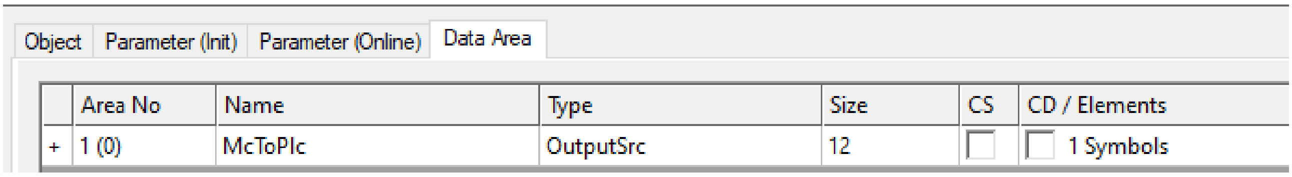
Parameter (Init): The group has no initial parameters.

Parameter (Online): The number of objects managed in the group (movers, tracks, environment) is displayed here. The state of the group is also displayed.



	Name	Online	CS	Type	PTC...	Comment
	Group object count	0	□	UDINT	0x0...	Number of objects in the group, read only.
	State	Disabled	□	MC.MC_PL...	0x0...	State, read only.

Data Area: Shows the memory area via which the group communicates with the PLC track.



	Area No	Name	Type	Size	CS	CD / Elements
+	1 (0)	McToPlc	OutputSrc	12	□	□ 1 Symbols

6.3.2 Example: "Creating and moving Planar movers with group"

Using this guide, you will create a TwinCAT project that contains two Planar movers and one Planar group. Both movers are added to the group and moved.

Creating a Planar mover

✓ See Configuration [▶ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
3. Change the start position of the second mover to x = 240.

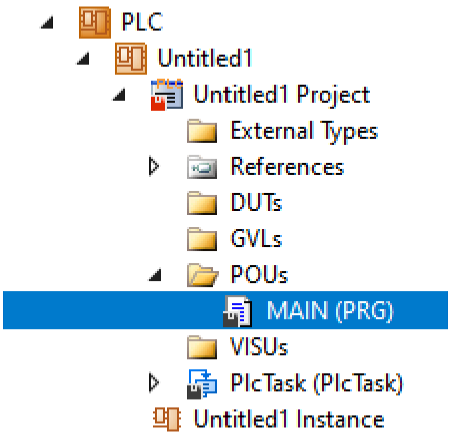
Creating a Planar group

4. Add the Planar group via **Groups > Add New Item…**, see Configuration [▶ 89].

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create two movers ("MC_PlanarMover") and a Planar group "MC_PlanarGroup" via **MAIN**.



⇨ These represent the movers and the group in the MC Configuration.

2. Create a state variable for a state machine as shown below, plus two auxiliary positions for the MoveToPosition [▶ 144] commands of the movers.

```
PROGRAM MAIN
VAR
    mover_one, mover_two : MC_PlanarMover;
    group : MC_PlanarGroup;
    state : UDINT;
    pos1, pos2 : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code activates the group and both movers. Both movers are then added to the group.

```
CASE state OF
  0:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 1;
  1:
    IF mover_one.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled
    AND mover_two.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
  2:
     group.Enable(0);
     state := 3;
  3:
    IF group.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
  4:
    mover_one.AddToGroup(0, group);
    mover_two.AddToGroup(0, group);
    state := 5;
  5:
    IF mover_one.MCTOPLC.STD.GroupOID = group.MCTOPLC_STD.GroupOID
    AND mover_two.MCTOPLC.STD.GroupOID = group.MCTOPLC_STD.GroupOID THEN
      state := 6;
    END_IF
  6:
    pos1.SetValuesXY(0, 240);
    pos2.SetValuesXY(0, 0);
    mover_one.MoveToPosition(0, pos1, 0, 0);
    mover_two.MoveToPosition(0, pos2, 0, 0);
    state := 7;
END_CASE
```

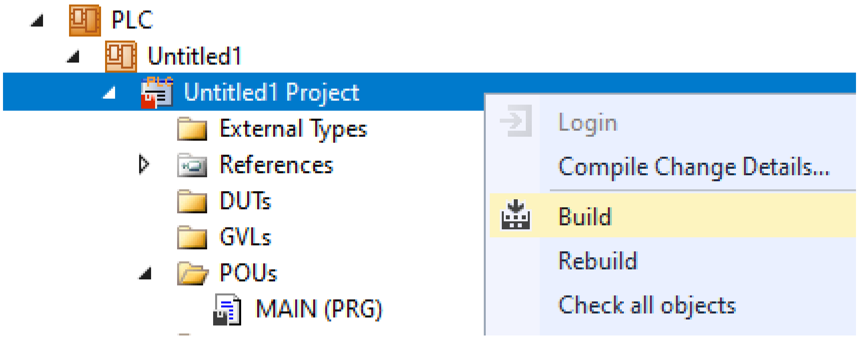
Sending the command

4. To send the command you must trigger the movers and the group cyclically using the update methods:

```
mover_one.Update();
mover_two.Update();
group.Update();
```

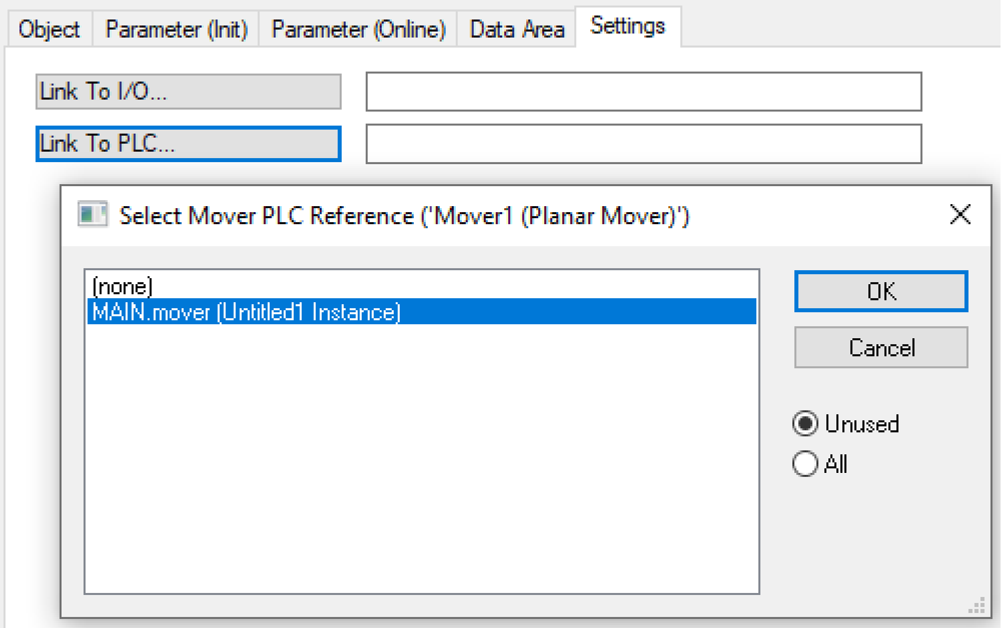
Building the PLC creates symbols of the "PLC mover" and the "PLC group", which can then be linked to the mover or group instance in the MC project.

5. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**

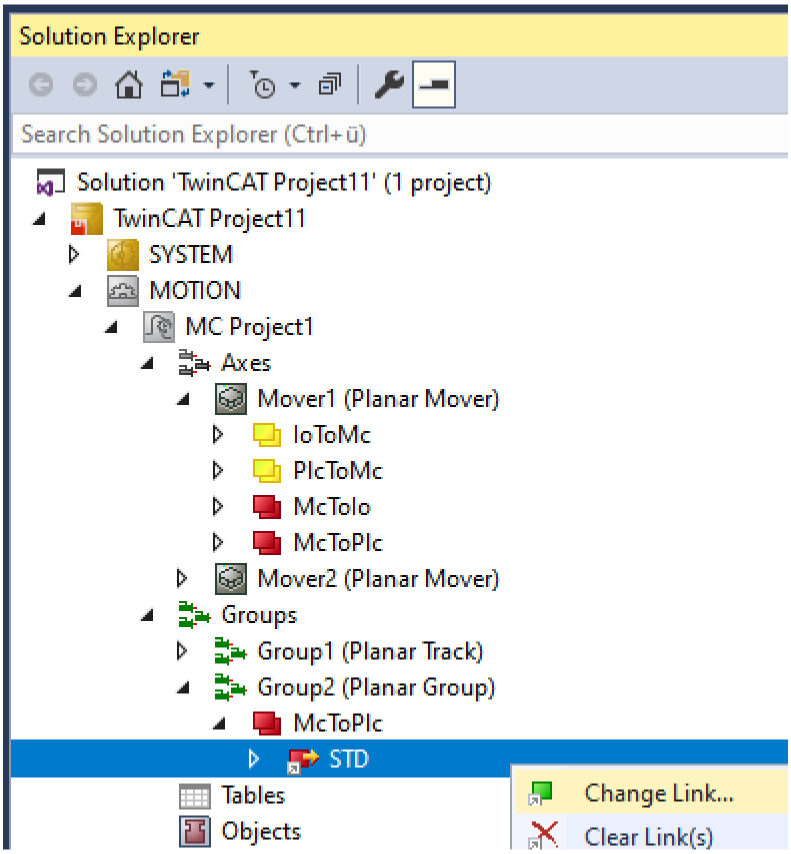


⇨ Subsequently, the Planar movers in the "MC Project" can be linked with the **Link To PLC...** button on the **Settings** tab.

6. Double-click Mover one first, then Mover two.



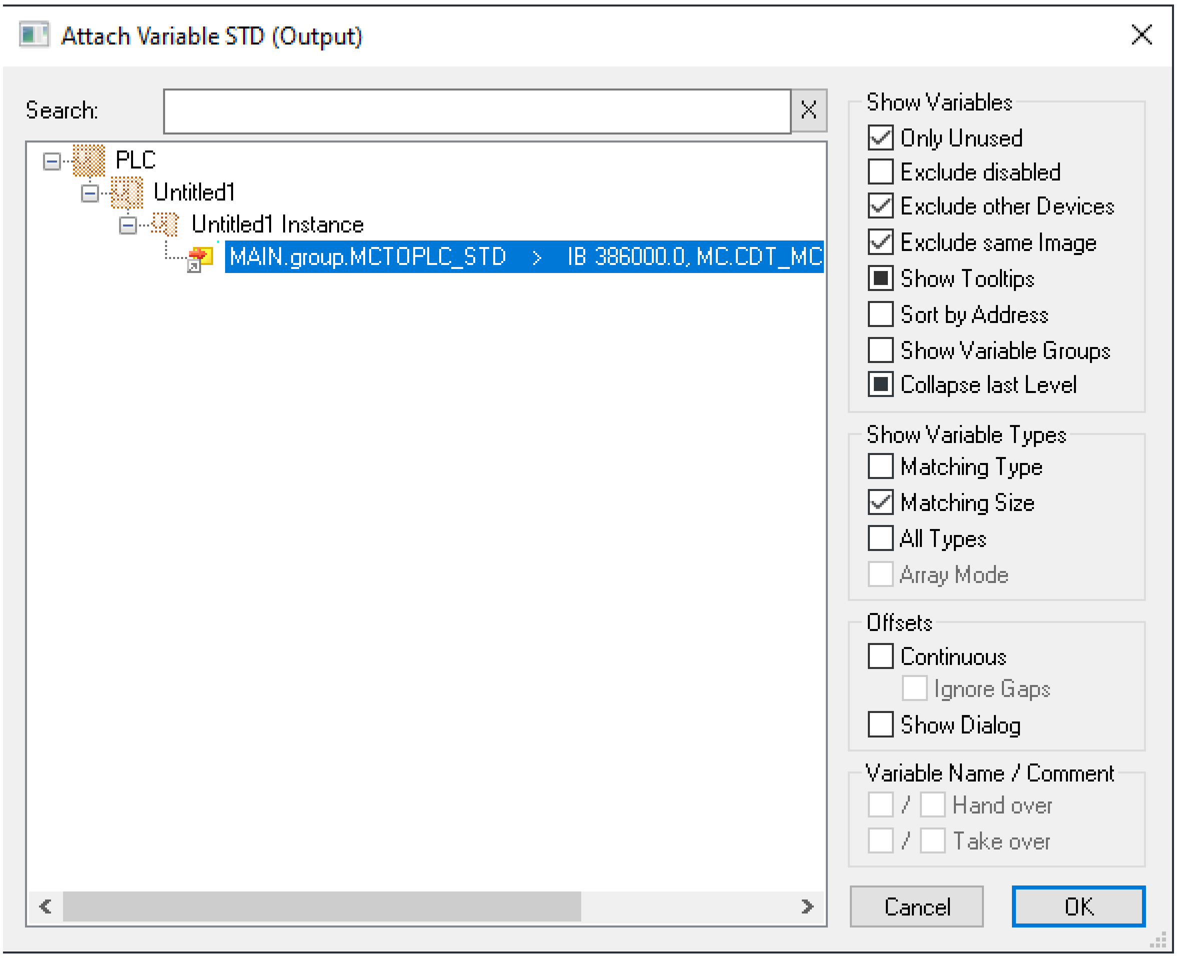
⇨ The group must be linked separately via the following dialog boxes.



Solution Explorer

Search Solution Explorer (Ctrl+ü)

- Solution 'TwinCAT Project11' (1 project)
  - TwinCAT Project11
    - SYSTEM
    - MOTION
      - MC Project1
        - Axes
          - Mover1 (Planar Mover)
            - IoToMc
            - PlcToMc
            - McToIo
            - McToPlc
          - Mover2 (Planar Mover)
        - Groups
          - Group1 (Planar Track)
          - Group2 (Planar Group)
            - McToPlc
              - STD
        - Tables
        - Objects



Attach Variable STD (Output)

Search:

- PLC
  - Untitled1
    - Untitled1 Instance
      - MAIN.group.MCTOPLC_STD > IB 386000.0, MC.CDT_MC

Show Variables
- ☑ Only Unused
- ☐ Exclude disabled
- ☑ Exclude other Devices
- ☑ Exclude same Image
- ■ Show Tooltips
- ☐ Sort by Address
- ☐ Show Variable Groups
- ■ Collapse last Level

Show Variable Types
- ☐ Matching Type
- ☑ Matching Size
- ☐ All Types
- ☐ Array Mode

Offsets
- ☐ Continuous
- ☐ Ignore Gaps
- ☐ Show Dialog

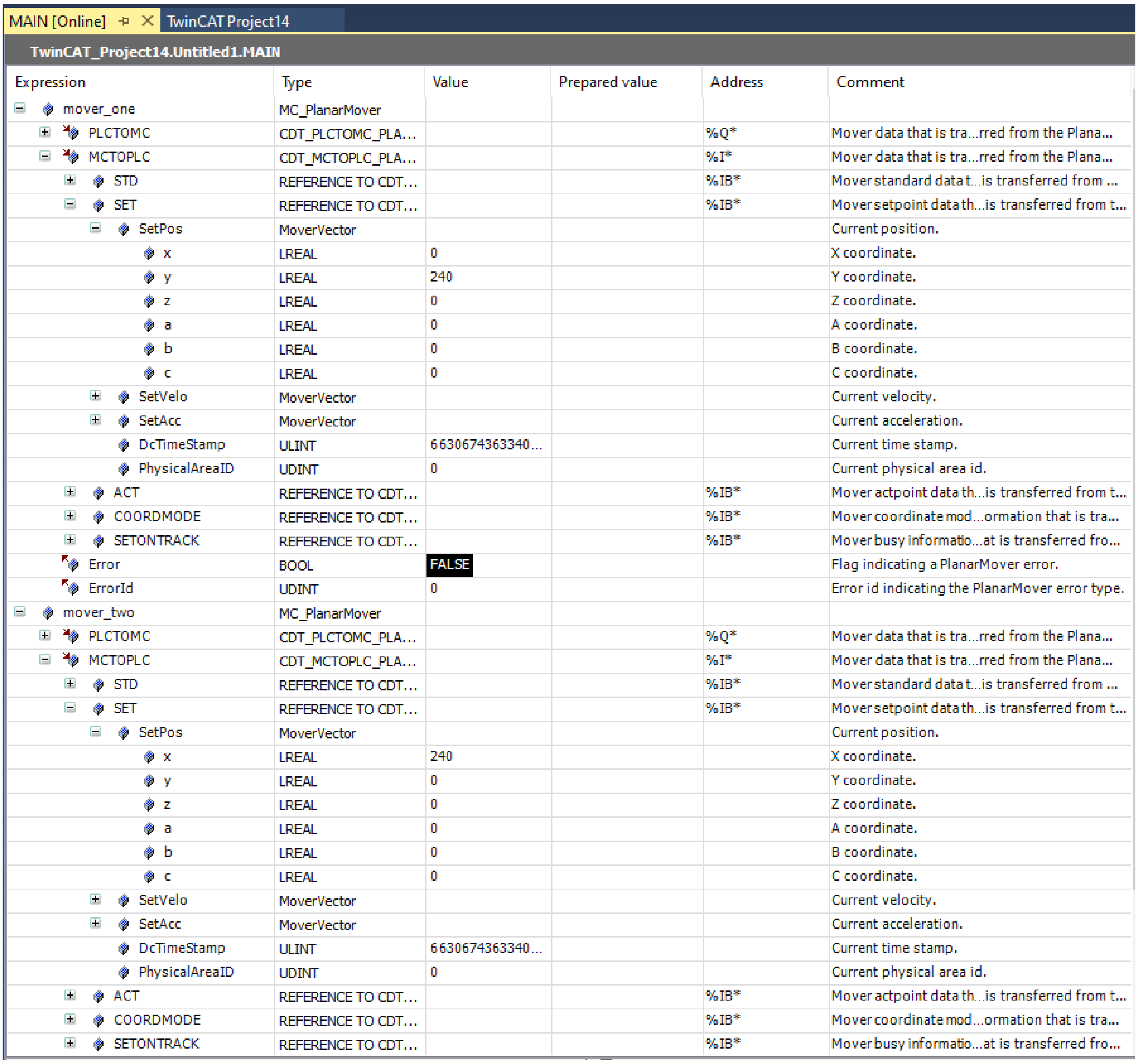
Variable Name / Comment
- ☐ / ☐ Hand over
- ☐ / ☐ Take over

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

After logging into the PLC and starting, you will see that the movers are not both in the target positions at the end of the state machine (state=7). Mover one has moved to x = 0 and y = 240. Mover two has not moved to the origin because Mover one still stood there and the command was therefore rejected because both are in a common group.

Since the dynamic limits of the movers are quite high by default, the change of positions after logging in may be difficult to follow with the naked eye. For the dynamic limits, see Planar mover [▶ 18].



TwinCAT_Project14.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
⊟ mover_one	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	0			X coordinate.
y	LREAL	240			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6630674363340...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
⊟ mover_two	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	240			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6630674363340...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT...			%IB*	Mover busy informatio...at is transferred fro...

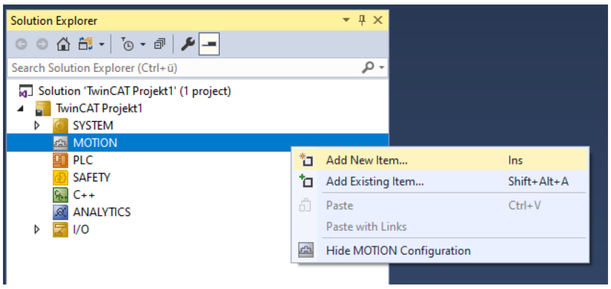
6.4 Planar environment

The Planar environment is a software object that represents the two-dimensional XPlanar stator surface. Together with Planar movers in a group, it prevents collisions of the movers with the edge of the surface.

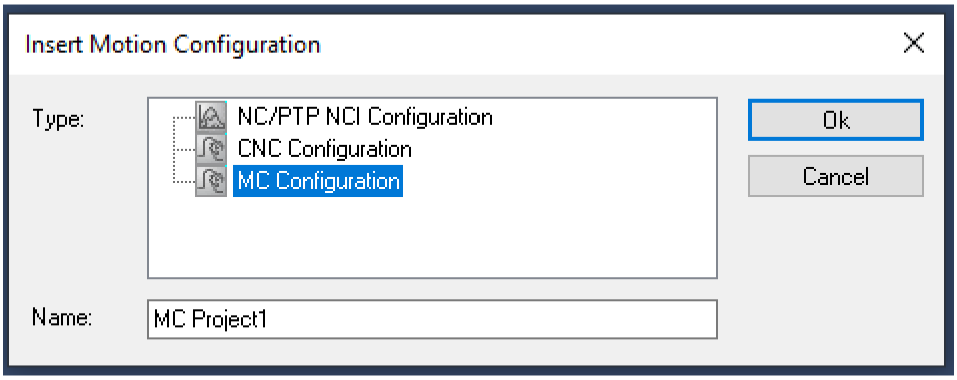
6.4.1 Configuration

✓ In order to create a Planar environment, an **MC Configuration** must first be created.

1. Select **MOTION** > **Add New Item…**.

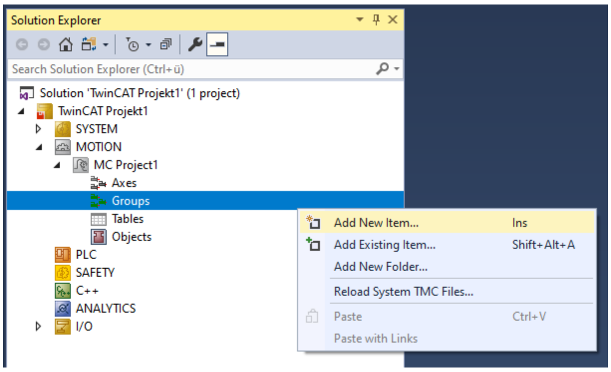

Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Projekt1' (1 project)
TwinCAT Projekt1
SYSTEM
MOTION
PLC
SAFETY
C++
ANALYTICS
I/O
Add New Item... Ins
Add Existing Item... Shift+Alt+A
Paste Ctrl+V
Paste with Links
Hide MOTION Configuration

2. In the following dialog box, select **MC Configuration** and confirm with **OK.**

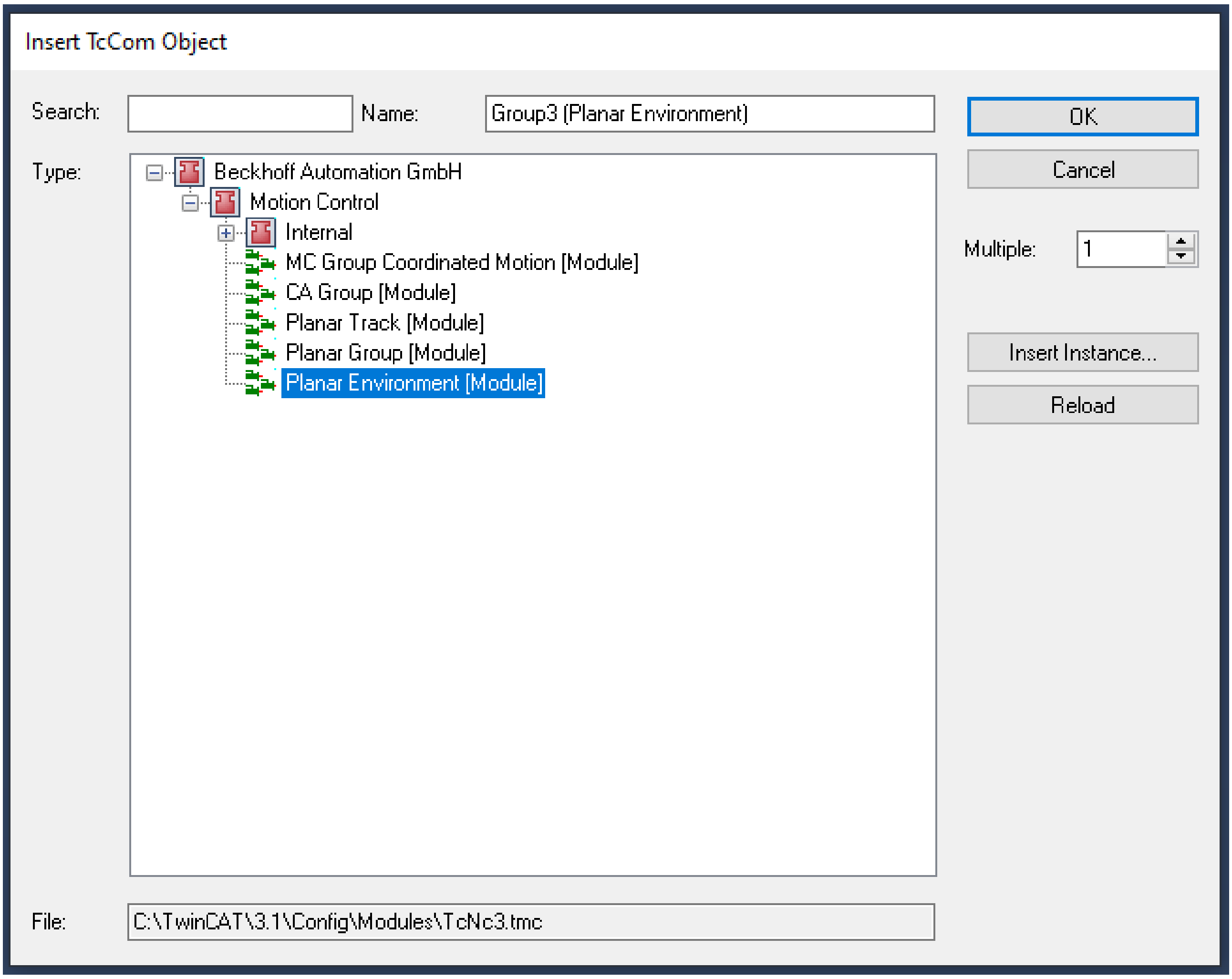


⇨ You have created an MC Project.

3. Select **MC Project** > **Groups** > **Add New Item…**.



4. In the following dialog box, create one (or more) Planar environments and confirm with**OK**.



Insert TcCom Object

Search:

Name: Group3 (Planar Environment)

Type:

- Beckhoff Automation GmbH
  - Motion Control
    - Internal
    - MC Group Coordinated Motion [Module]
    - CA Group [Module]
    - Planar Track [Module]
    - Planar Group [Module]
    - Planar Environment [Module]

Multiple: 1

File: C:\TwinCAT\3.1\Config\Modules\TcNc3.tmc

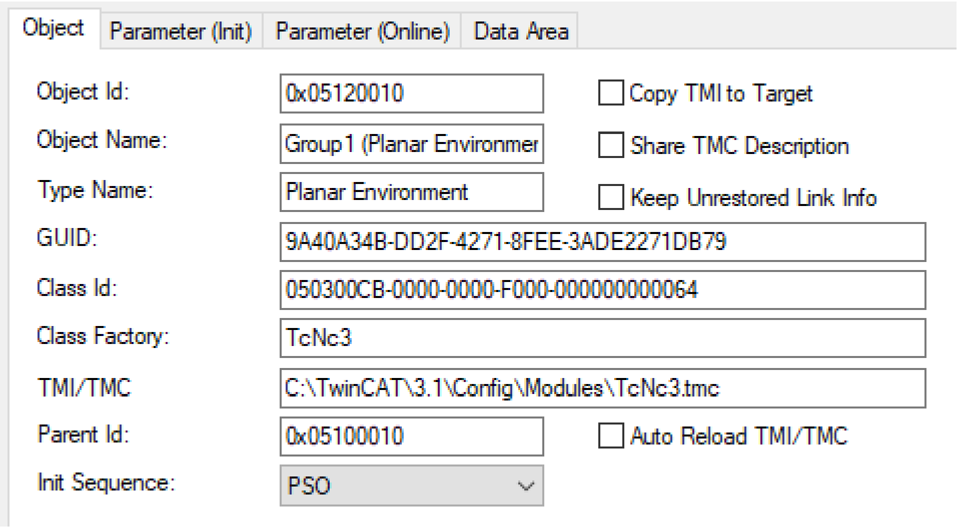
⇨ The Planar environment is now created and can be parameterized.

Open detailed description

- Select the Planar environment in the tree and double-click it.

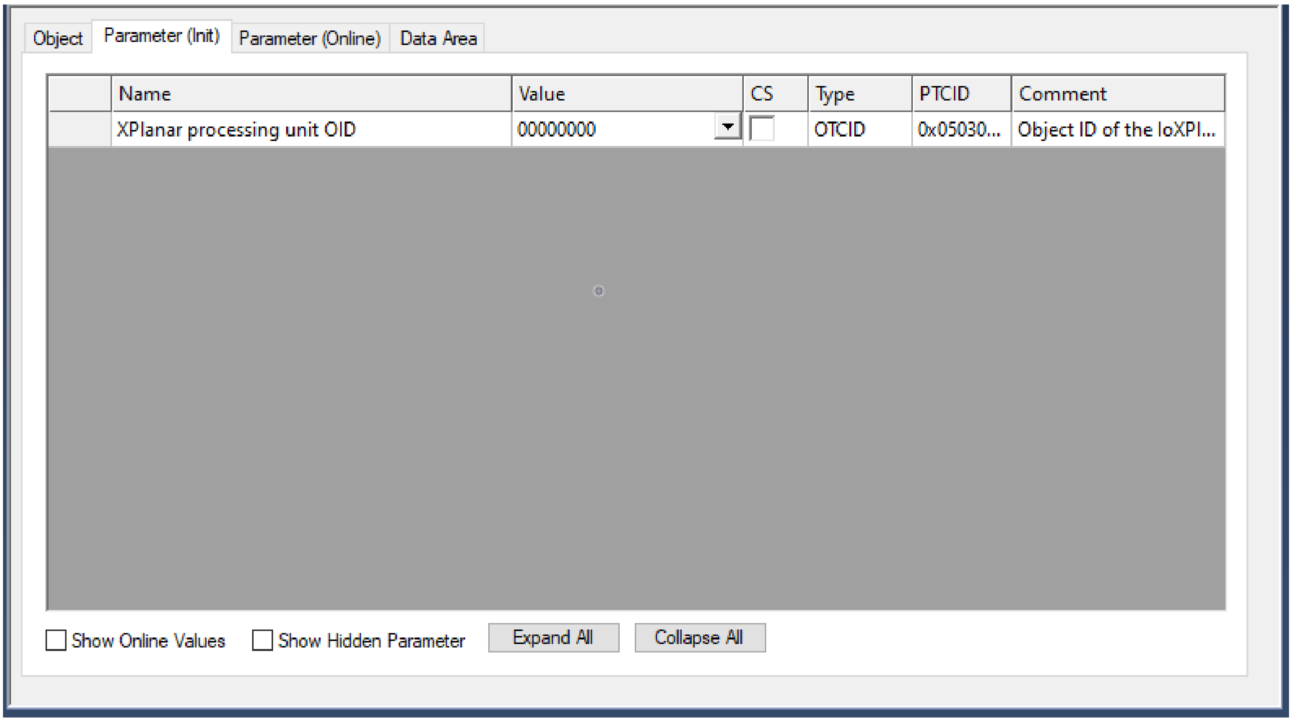
Purposes of the individual tabs

Object: General information (name, type, ID and so on) is shown here.



Object Id:	0x05120010	☐ Copy TMI to Target
Object Name:	Group1 (Planar Environmer	☐ Share TMC Description
Type Name:	Planar Environment	☐ Keep Unrestored Link Info
GUID:	9A40A34B-DD2F-4271-8FEE-3ADE2271DB79	
Class Id:	050300CB-0000-0000-F000-000000000064	
Class Factory:	TcNc3	
TMI/TMC	C:\TwinCAT\3.1\Config\Modules\TcNc3.tmc	
Parent Id:	0x05100010	☐ Auto Reload TMI/TMC
Init Sequence:	PSO	

Parameter (Init): Specifies initial parameters that the user can change in order to affect the behavior of the environment.

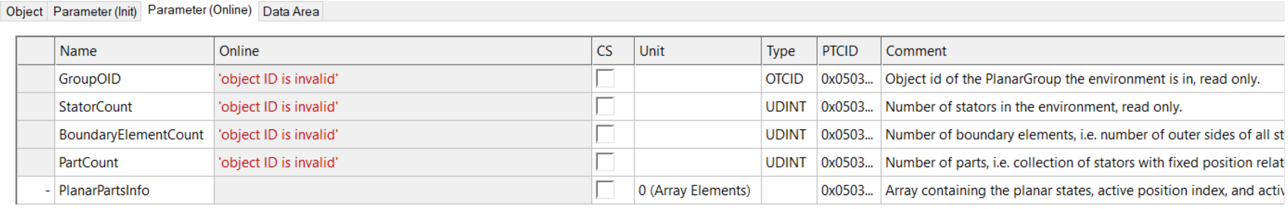


	Name	Value	CS	Type	PTCID	Comment
	XPlanar processing unit OID	00000000	□	OTCID	0x05030...	Object ID of the IoXPl...

The environment has the initial parameter "XPlanar processing unit OID". When this (>0) is set to the object ID of the XPlanar processing unit, the environment automatically reads the stator configuration from the XPlanar processing unit and generates the boundary elements for collision detection from this information. This takes place as soon as the user calls the CreateBoundary() command in the PLC.

From version V3.2.60: If the "XPlanar processing unit OID" parameter is set to the object ID of the XPlanar processing unit, the environment also reads the part configuration from the XPlanar processing unit and generates an internal representation of all parts. This is used both to perform collision checks with the edge of the parts when the environment is in the Planar group and to provide all components (movers, tracks, group) with a complete system description.

Parameter (Online): Shows the state of the environment during the runtime of the object.

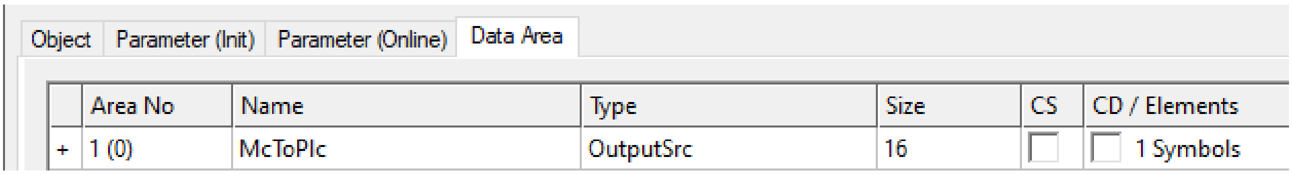


	Name	Online	CS	Unit	Type	PTCID	Comment
	GroupOID	'object ID is invalid'	□		OTCID	0x0503...	Object id of the PlanarGroup the environment is in, read only.
	StatorCount	'object ID is invalid'	□		UDINT	0x0503...	Number of stators in the environment, read only.
	BoundaryElementCount	'object ID is invalid'	□		UDINT	0x0503...	Number of boundary elements, i.e. number of outer sides of all st
	PartCount	'object ID is invalid'	□		UDINT	0x0503...	Number of parts, i.e. collection of stators with fixed position relat
-	PlanarPartsInfo		□	0 (Array Elements)		0x0503...	Array containing the planar states, active position index, and activ

The number of stators inserted into the environment and the boundary elements calculated from them are displayed here.

From version V3.2.60: The "PartCount" parameter specifies the number of parts read out and created internally. The "PlanarPartsInfo" parameter displays information for all parts. This information consists of the object ID of the part, the Planar state of the part, the active position index, the position of the part consisting of the object ID of the coordinate system and x/y coordinates, and the "disableForced" flag of the part.

Data Area: Shows the memory area via which the group communicates with the PLC environment.



	Area No	Name	Type	Size	CS	CD / Elements
+	1 (0)	McToPlc	OutputSrc	16	□	□ 1 Symbols

6.4.2 Example "Configuring the stator area and boundary"

Using this guide you will be able to create a TwinCAT project that contains a Planar environment and you will configure its stator surface and boundary.

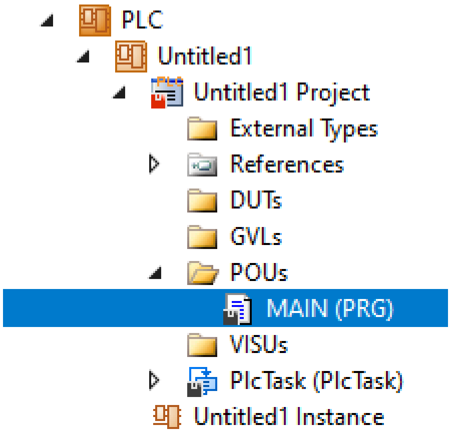
Creating a Planar environment

1. Create a Planar environment, see Configuration [▶ 96].

Creating a PLC

✓ See preliminary steps Creating a PLC [▶ 21].

1. Create an "MC_PlanarEnvironment" via **MAIN**.



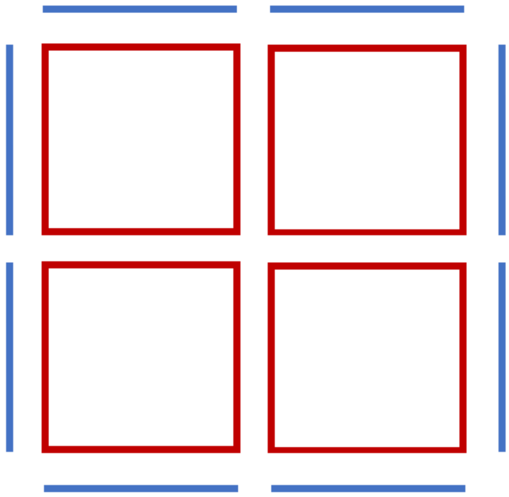
⇨ This represents the environment in the MC configuration.

2. Create a state variable for a state machine as shown below.

```
PROGRAM MAIN
VAR
    environment : MC_PlanarEnvironment;
    state : UDINT;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code adds four stators to the environment. The lower left corner of the square stators (side length 240 mm) is specified in each case. CreateBoundary() then calculates the outer boundary of the stator surface.
The stators (red) and boundary elements (blue) are shown schematically in the following illustration.



```
CASE state OF
  0:
    environment.AddStator(0,0.0,0.0);
    environment.AddStator(0,240.0,0.0);
    environment.AddStator(0,0.0,240.0);
    environment.AddStator(0,240.0,240.0);
    environment.CreateBoundary(0);
    state := 1;
END_CASE
```

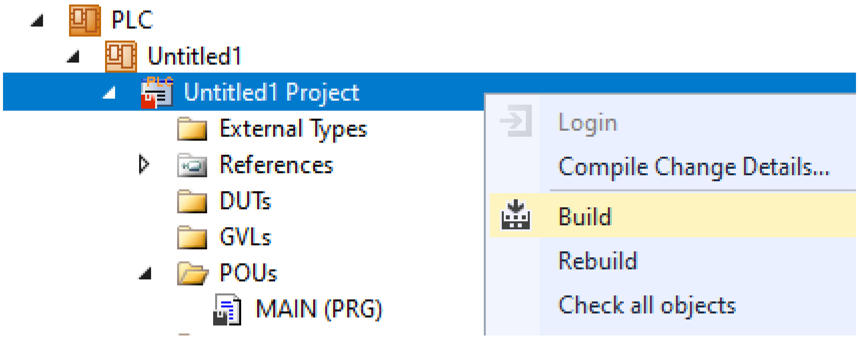
Sending the command

4. To send the command, you must call the environment cyclically with its update method after the END_CASE:

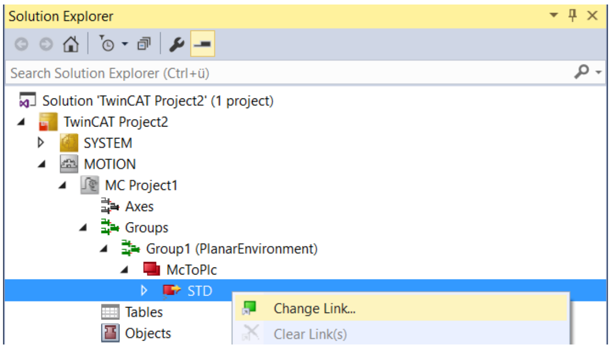
```
environment.Update();
```

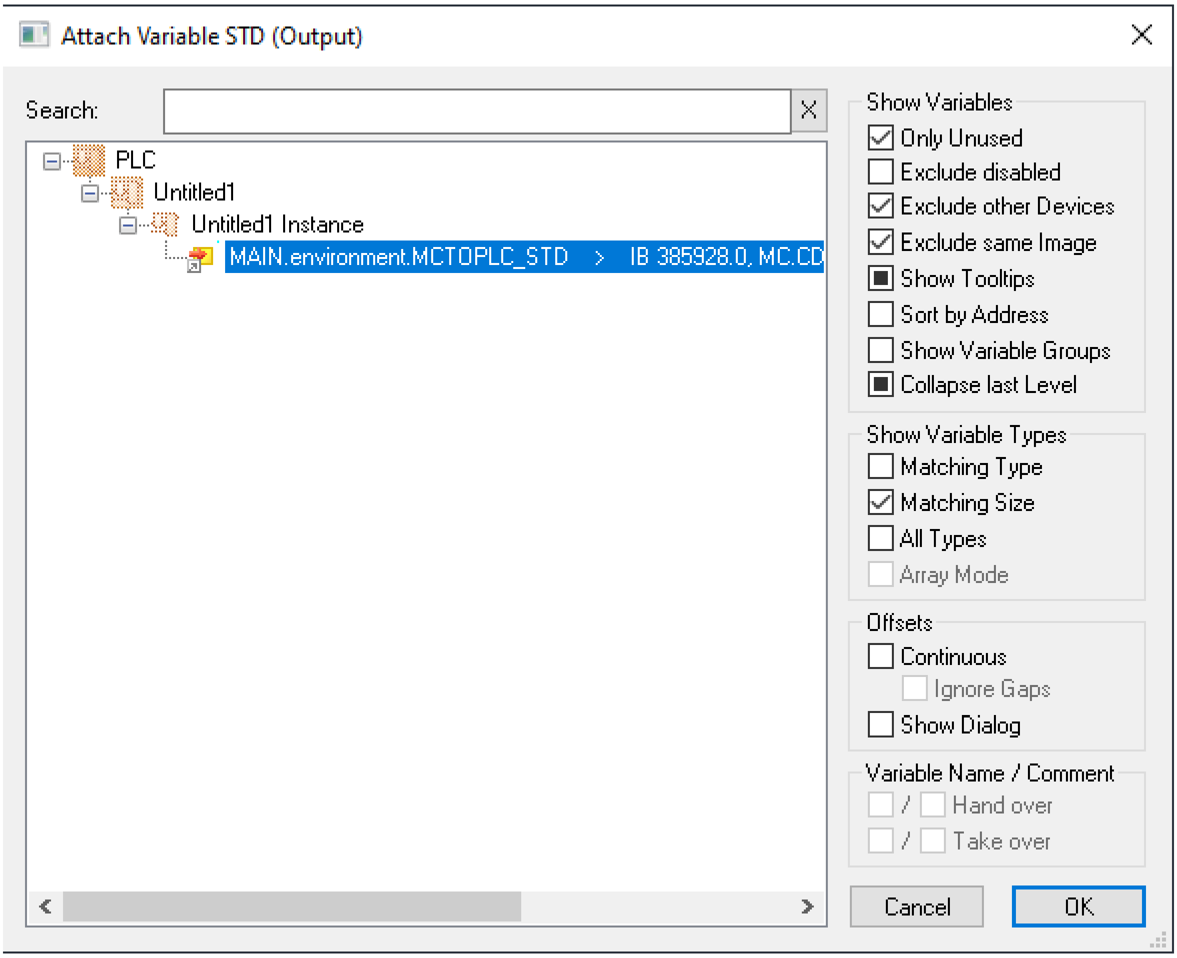
When creating the PLC, a symbol of the "PLC environment" is created, which can then be linked to the Planar environment in the MC project.

5. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ The Planar environment can then be linked in the "MC Project".





Attach Variable STD (Output)

Search:

- PLC
  - Untitled1
    - Untitled1 Instance
      - MAIN.environment.MCTOPLC_STD > IB 385928.0, MC.CD

Show Variables
- ☑ Only Unused
- □ Exclude disabled
- ☑ Exclude other Devices
- ☑ Exclude same Image
- ■ Show Tooltips
- □ Sort by Address
- □ Show Variable Groups
- ■ Collapse last Level

Show Variable Types
- □ Matching Type
- ☑ Matching Size
- □ All Types
- □ Array Mode

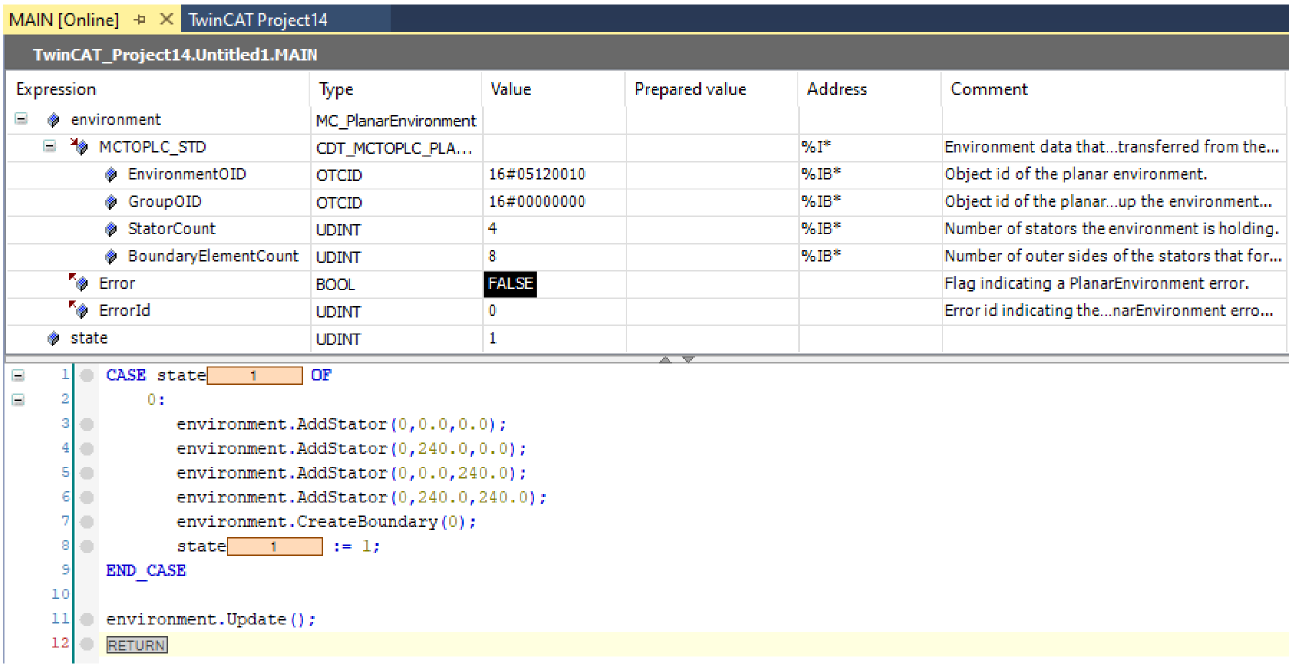
Offsets
- □ Continuous
  - □ Ignore Gaps
- □ Show Dialog

Variable Name / Comment
- □ / □ Hand over
- □ / □ Take over

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

The environment is in the desired state at the end of the state machine (state = 1).



Expression	Type	Value	Prepared value	Address	Comment
environment	MC_PlanarEnvironment				
MCTOPLC_STD	CDT_MCTOPLC_PLA...			%I*	Environment data that...transferred from the...
EnvironmentOID	OTCID	16#05120010		%IB*	Object id of the planar environment.
GroupOID	OTCID	16#00000000		%IB*	Object id of the planar...up the environment...
StatorCount	UDINT	4		%IB*	Number of stators the environment is holding.
BoundaryElementCount	UDINT	8		%IB*	Number of outer sides of the stators that for...
Error	BOOL	FALSE			Flag indicating a PlanarEnvironment error.
ErrorId	UDINT	0			Error id indicating the...narEnvironment erro...
state	UDINT	1			

```
CASE state 1 OF
    0:
        environment.AddStator(0,0.0,0.0);
        environment.AddStator(0,240.0,0.0);
        environment.AddStator(0,0.0,240.0);
        environment.AddStator(0,240.0,240.0);
        environment.CreateBoundary(0);
        state 1 := 1;
END_CASE

environment.Update();
RETURN
```

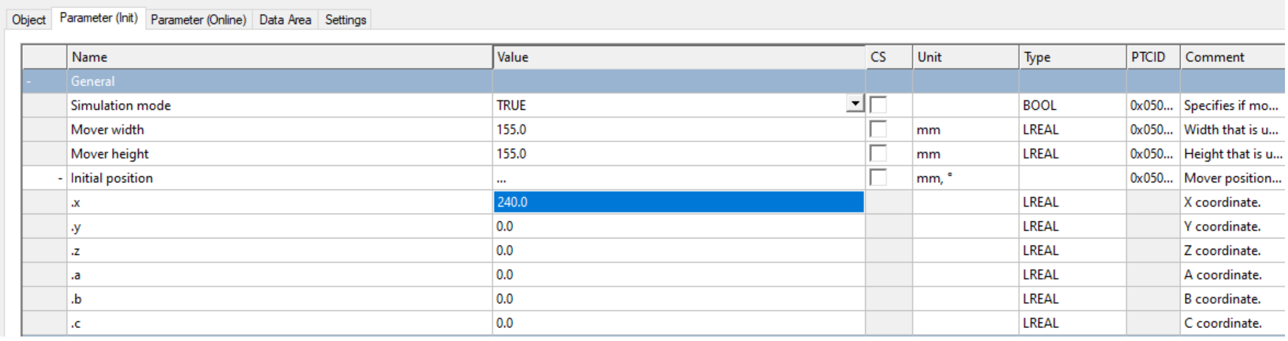
6.5 Example: "Creating and moving Planar movers with track and group"

Using this guide you will create a TwinCAT project that includes two Planar movers, a Planar track and a Planar group, and moves the movers both on and alongside the track.

Creating a Planar mover

✓ See Configuration [▸ 18].

1. Create two Planar movers.
2. Put "Parameter (Init)" into simulation mode (TRUE). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
3. Change the start position of the second mover to x = 240.



Name	Value	CS	Unit	Type	PTCID	Comment
General						
Simulation mode	TRUE	□		BOOL	0x050...	Specifies if mo...
Mover width	155.0	□	mm	LREAL	0x050...	Width that is u...
Mover height	155.0	□	mm	LREAL	0x050...	Height that is u...
Initial position	...	□	mm, °		0x050...	Mover position...
.x	240.0			LREAL		X coordinate.
.y	0.0			LREAL		Y coordinate.
.z	0.0			LREAL		Z coordinate.
.a	0.0			LREAL		A coordinate.
.b	0.0			LREAL		B coordinate.
.c	0.0			LREAL		C coordinate.

Creating a Planar track and Planar group

4. Add the Planar track via **Groups > Add New Item…**, see Configuration [▸ 42].
5. Proceed in the same way for the Planar group.

Creating a PLC

✓ To control the movers, the track and the group, a PLC must be created from which the user can issue commands to the mover, see Creating a PLC [▸ 21].

6. Create two mowers (MC_PlanarMovers [▸ 143]), an MC_PlanarTrack [▸ 161] and an MC_PlanarGroup [▸ 141] via MAIN.

⇨ These represent the movers, the track and the group in the MC Configuration.

7. Create a state variable for a state machine and two auxiliary positions for the track, as shown below.
8. Also create a feedback.

   ⇨ The feedback can be associated with any commands. It provides detailed information about the command execution and the execution time.

```
PROGRAM MAIN
VAR
    mover_one, mover_two : MC_PlanarMover;
    track : MC_PlanarTrack;
    group : MC_PlanarGroup;
    state : UDINT;
    pos1, pos2 : PositionXYC;
    feedback : MC_PlanarFeedback;
END_VAR
```

9. Then program a sequence in MAIN.

   ⇨ This program code creates and activates a track, a group and both movers. Both the movers and the track are added to the group. After that, Mover one is joined and moved on the track. When moving, feedback is provided via which we receive the rejection of the command as an error. The command is rejected because Mover two is blocking the track (collision error).

```
CASE state OF
  0:
    pos1.SetValuesXY(0, 0);
    pos2.SetValuesXY(400, 0);
    track.AppendLine(0, pos1, pos2);
    track.Enable(0);
    group.Enable(0);
    state := 1;
  1:
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled
    AND group.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
         state := 2;
    END_IF
  2:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 3;
  3:
    IF mover_one.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled
    AND mover_two.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
         state := 4;
    END_IF
  4:
    mover_one.AddToGroup(0,group);
    mover_two.AddToGroup(0,group);
    track.AddToGroup(0,group);
    state := 5;
  5:
    IF mover_one.MCTOPLC.STD.GroupOID > 0
    AND mover_two.MCTOPLC.STD.GroupOID > 0
    AND track.MCTOPLC_STD.GroupOID > 0 THEN
         state := 6;
    END_IF
  6:
    mover_one.JoinTrack(0, track, 0, 0);
    state := 7;
  7:
    IF mover_one.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
         state := 8;
    END_IF
  8:
    mover_one.MoveOnTrack(feedback, 0, 150.0, 0, 0);
    pos2.SetValuesXY(240, 320);
    mover_two.MoveToPosition(0, pos2, 0, 0);
    state := 9;
  9:
    IF NOT mover_two.MCTOPLC.STD.Busy.busyXYC THEN
         state := 10;
    END_IF
  10:
    mover_one.MoveOnTrack(0, 0, 150.0, 0, 0);
    state := 11;
  11:
    IF NOT mover_one.MCTOPLC.STD.Busy.busyXYC THEN
         state := 12;
    END_IF

END_CASE
```

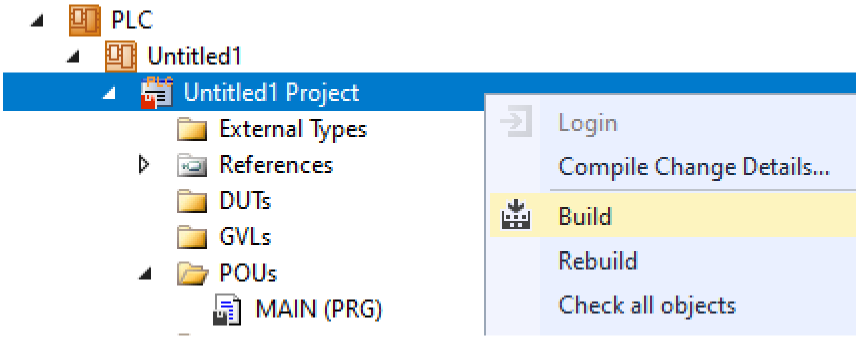
Sending the command

10. To send the command, you must call the mover, the track and the group cyclically with their update method after the END_CASE:

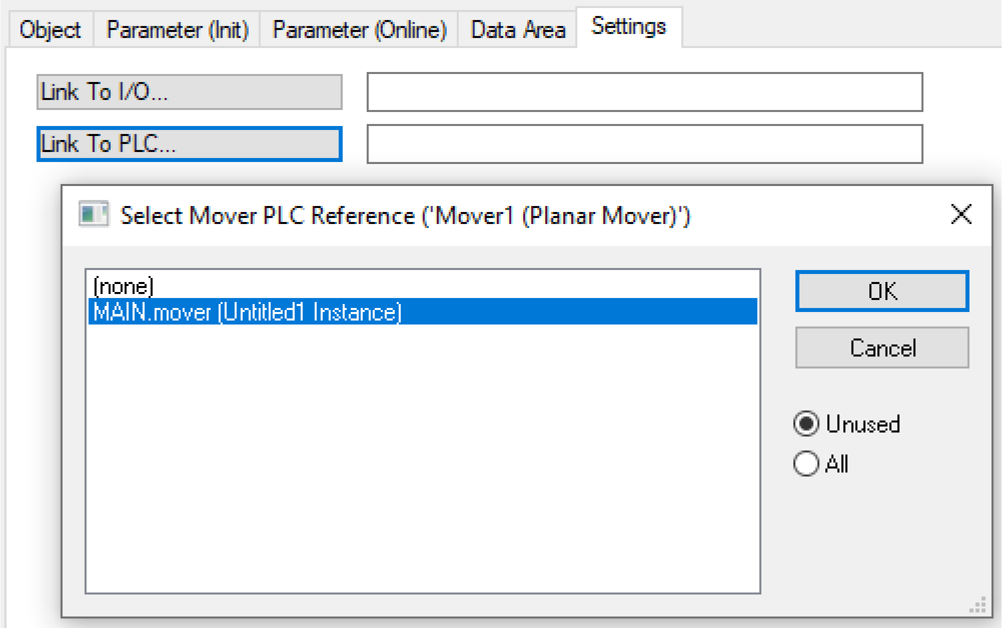
```
mover_one.Update();
mover_two.Update();
track.Update();
group.Update();
feedback.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.

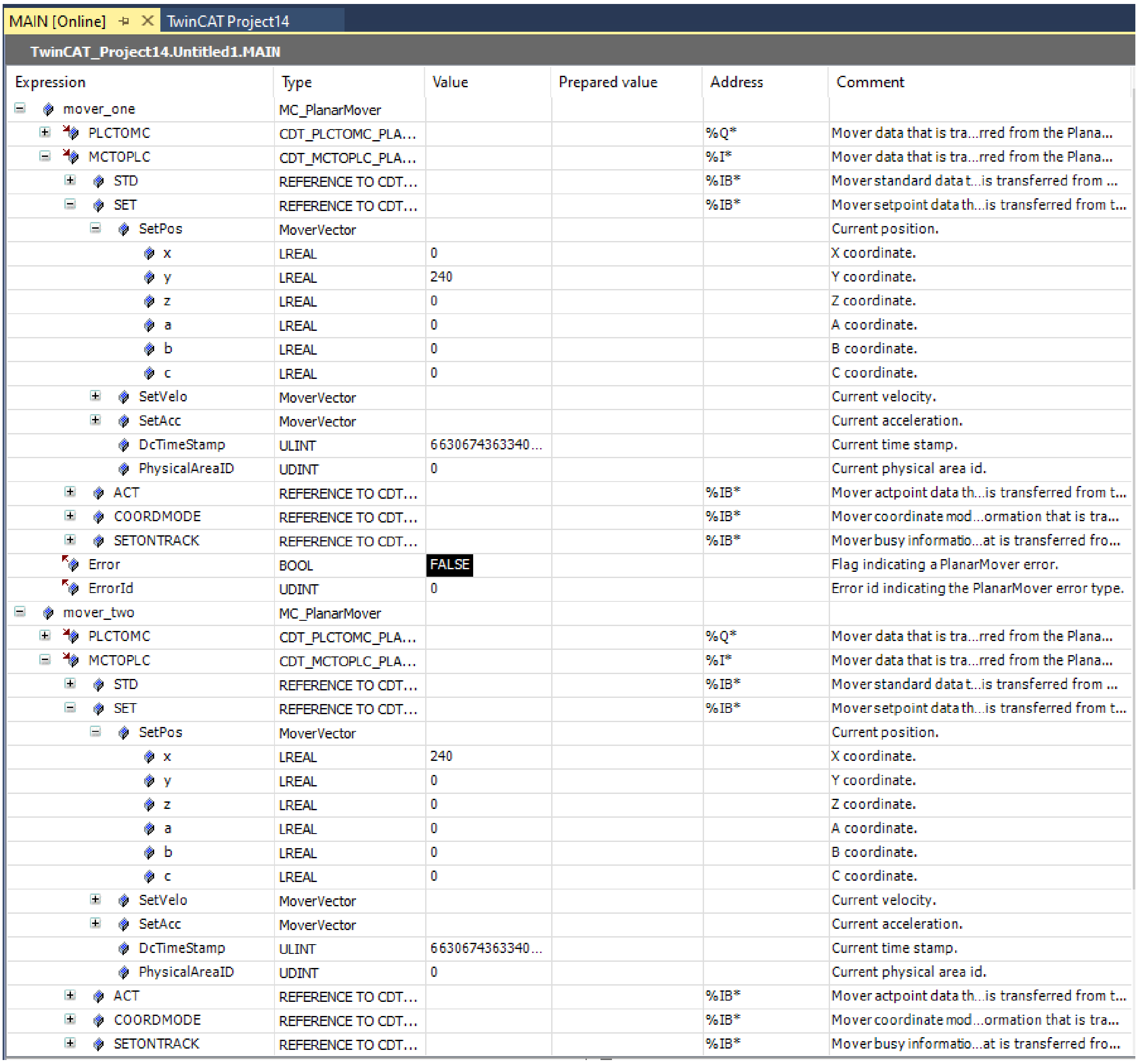


Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=12), the movers are in the desired position.

The feedback indicates the collision error. In addition, in case of collision errors in the feedback, the blocking object is displayed with its OID. It would now be possible, after Mover two has been moved out of the way, to move Mover one on the track.



Expression	Type	Value	Prepared value	Address	Comment
⊟ mover_one	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA…			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA…			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT…			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT…			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	0			X coordinate.
y	LREAL	240			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6630674363340…			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT…			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT…			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT…			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
⊟ mover_two	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA…			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA…			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT…			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT…			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	240			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6630674363340…			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT…			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT…			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT…			%IB*	Mover busy informatio...at is transferred fro...

6.6 Planar part

From version V3.2.60: The Part feature, which is the subject of this section, is available.

The MC_PlanarPart [▸ 158] is a PLC software object that represents the part in the PLC. It displays the state of the part and offers methods for changing the status.

The connection of the MC_PlanarPart [▸ 158] is established indirectly via the MC_PlanarEnvironment [▸ 134]. To do this, the initialize method of the MC_PlanarPart [▸ 158] must be called with the correct object ID of the part after starting the PLC. The state of the MC_PlanarPart [▸ 158] can then be both read and changed by method calls.

Possible commands for the MC_PlanarPart [▸ 158] are the ActivatePosition [▸ 159] method and the AllowEnable [▸ 160], ForceDisable [▸ 160] and Reset [▸ 160] methods. The ActivatePosition [▸ 159] method moves the part to one of the possible positions. The methods AllowEnable [▸ 160], ForceDisable [▸ 160] and Reset [▸ 160] change the PlanarState of the MC_PlanarPart [▸ 158].

After calling the AllowEnable [▶ 160] method, the PlanarPart is allowed to start up the PlanarState Machine (up to the CoE state OperationEnabled). As soon as the first mover in a coordinate system is activated with an Enable() command, all parts in this system are activated. Activating the part that the mover is located on is necessary for activating the mover and takes place first. Activating the remaining parts in a coordinate system is optional and may fail without preventing or canceling the activation. Conversely, calling the Disable() command on the last active mover in a coordinate system causes all parts to be switched off. The parts do not have to be switched off in order to switch off the mover and this takes place after the mover has been switched off.

Similarly, parts are activated or deactivated when the first/last activated mover enters or leaves a coordinate system by calling the ActivatePosition [▶ 159] method of its part. The parts are also deactivated if the last active mover has an error with an "abort" error response. If a mover has an error with the "quick stop" error response, the parts remain activated. This applies during the error and the subsequent reset until the part is reactivated. If the error response is changed to "abort" by a Disable() command from the mover during the error or the Reset [▶ 160] command, the parts switch off.

If a part is forced into the disabled state by calling the ForceDisable [▶ 160] method, all movers on this part are set to the error state and other parts in the coordinate system may be switched off if there are no other active movers in the coordinate system. The part remains in the disabled state until at least the next AllowEnable [▶ 160] call.

Error states of the part force errors of all movers on this part. The errors of the movers are attained before the part error. Other movers or parts in the coordinate system are not affected by these errors.

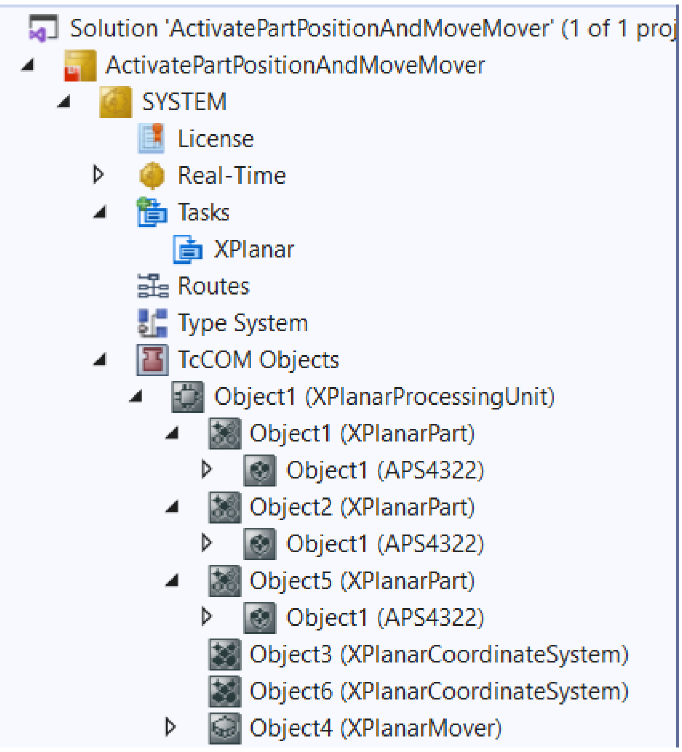
In the event of a part error, a Reset [▶ 160] command can be triggered to rectify the error. Any mover errors are not affected by this. Conversely, a Reset [▶ 160] command from a mover triggers a Reset [▶ 160] command for all parts in the coordinate system with an error. The Reset [▶ 160] command of the mover has as a necessary condition that its own part is error-free; accordingly, the part is reset before the mover.

6.6.1 Example "Activating a Planar part position and moving a Planar mover"

In this example, a Planar mover is moved to three Planar parts while the movable one of the parts is moved.

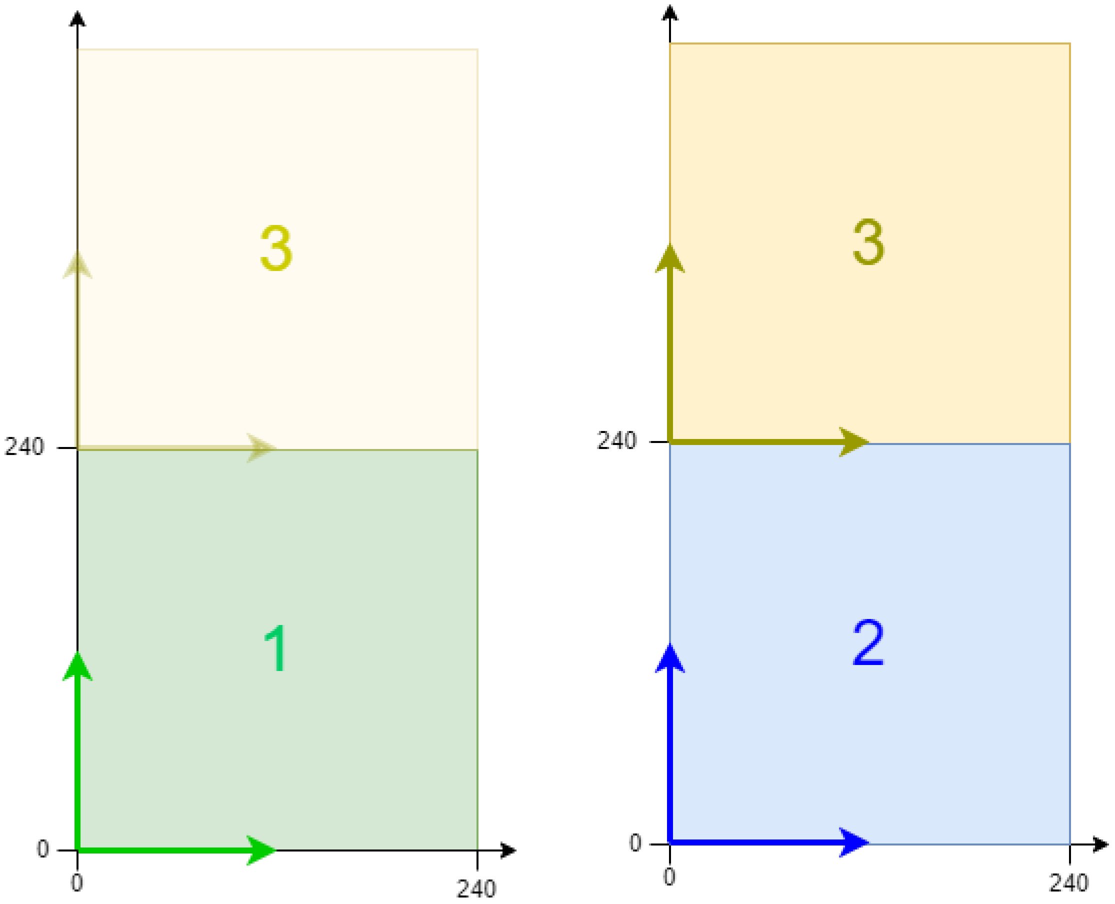
Starting point

You start with a solution that contains a fully configured XPlanar Processing Unit. Three parts, two coordinate systems and a mover are created under the XPlanar Processing Unit. A tile is created under each of the three parts.



Solution 'ActivatePartPositionAndMoveMover' (1 of 1 proj
ActivatePartPositionAndMoveMover
SYSTEM
License
Real-Time
Tasks
XPlanar
Routes
Type System
TcCOM Objects
Object1 (XPlanarProcessingUnit)
Object1 (XPlanarPart)
Object1 (APS4322)
Object2 (XPlanarPart)
Object1 (APS4322)
Object5 (XPlanarPart)
Object1 (APS4322)
Object3 (XPlanarCoordinateSystem)
Object6 (XPlanarCoordinateSystem)
Object4 (XPlanarMover)

The following geometric situation is set: the first two parts are fixed in the two coordinate systems at the origin and the third part can change position between the two coordinate systems. It is conceivable, for example, that the two coordinate systems are arranged one above the other in two planes and Part 3 is an elevator between the two systems. The mover starts in the middle of the first part in coordinate system 1, while the third part starts in coordinate system 2.



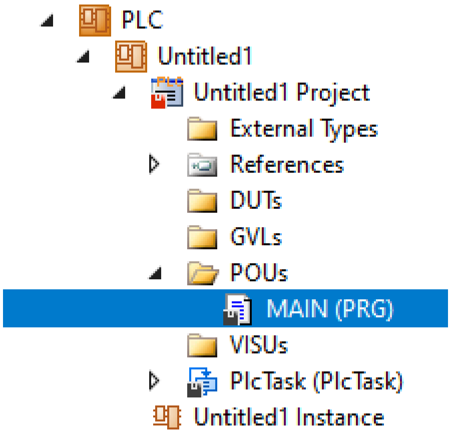
Creating a Planar mover and a Planar environment

1. Create a Planar mover for this example, see Configuration [▸ 18].
2. Create a Planar environment, see Configuration [▸ 96].
3. Set the initial parameter XPlanar processing unit OID to the object ID of the XPlanar Processing Unit. This activates the Part feature for all **MC Configuration** objects (especially for the created Planar mover).

Creating a PLC

✓ See preliminary steps Creating a PLC [▸ 21].

1. Use **MAIN** to create the mover(s) ("MC_PlanarMover [▸ 143]") as follows.



⇨ This/these represent(s) the mover(s) in the MC Configuration.

2. Create a Planar mover, a Planar environment, a Planar part, a state variable for a state machine and a target position for a travel command of the mover, as shown below.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    environment : MC_PlanarEnvironment;
    part_three : MC_PlanarPart;
    state : UDINT;
    target_position : PositionXYC;
END_VAR
```

3. Then program a sequence in MAIN.

⇨ This program code initializes part 3, activates the mover, moves part 3 to coordinate system 1, moves the mover to part 3, moves part 3 back to coordinate system 2 and finally moves the mover to part 2 in coordinate system 2.

```
CASE state OF
  0:
    part_three.Initialize(0, 16#01010080, environment);
    state := 1;
  1:
    IF part_three.IsInitialized THEN
      state := 2;
    END_IF
  2:
    mover.Enable(0);
    state := 3;
  3:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
  4:
    part_three.ActivatePosition(0,1);
    state := 5;
  5:
    IF part_three.PositionIndex = 1 THEN
      state := 6;
    END_IF
  6:
    target_position.SetValuesXYCReferenceId(120, 120, 0, part_three.PartOID);
    mover.MoveToPosition(0, target_position, 0, 0);
    state := 7;
  7:
    IF mover.MCTOPLC.SET.SetPos.y > 300 AND NOT mover.MCTOPLC.STD.Busy.busyXYC THEN
      state := 8;
    END_IF
  8:
    part_three.ActivatePosition(0,2);
    state := 9;
  9:
    IF part_three.PositionIndex = 2 THEN
      state := 10;
    END_IF
  10:
    target_position.SetValuesXYCReferenceId(120, 120, 0, 16#01010030); // Position on part two
    mover.MoveToPosition(0, target_position, 0, 0);
    state := 11;

END_CASE
```

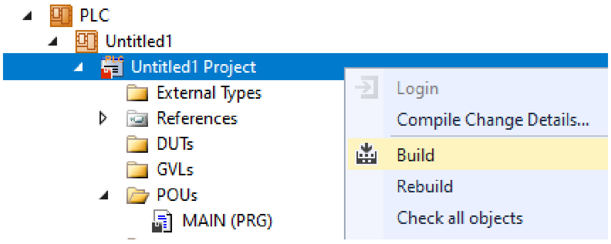
Sending the command

4. To send the motion command, you must call the mover cyclically with its update method after the END_CASE; to send the commands of the Planar part, the environment must be called cyclically with its update method:

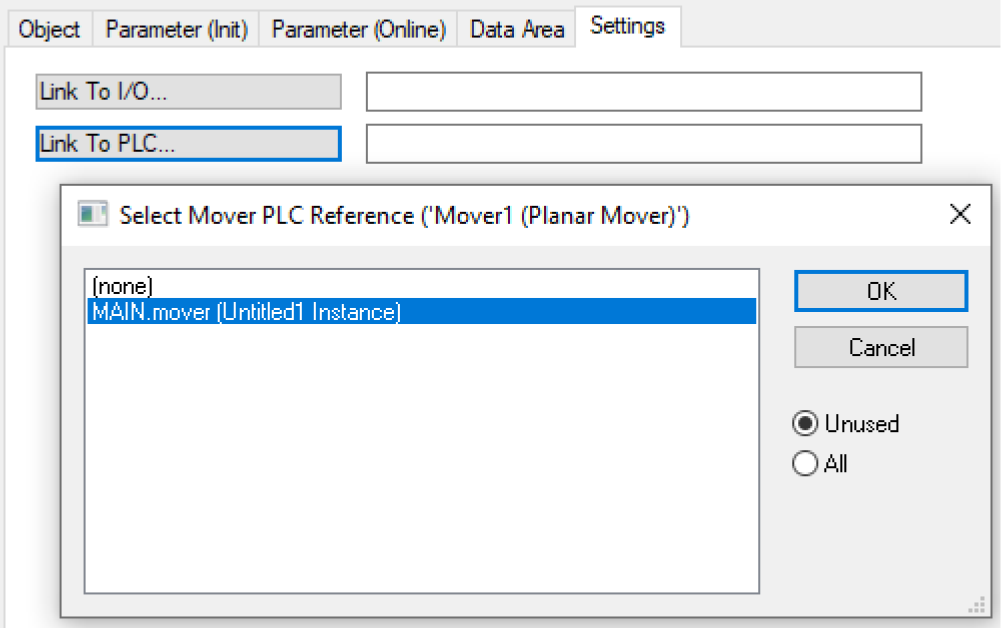
```
mover.Update();
environment.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

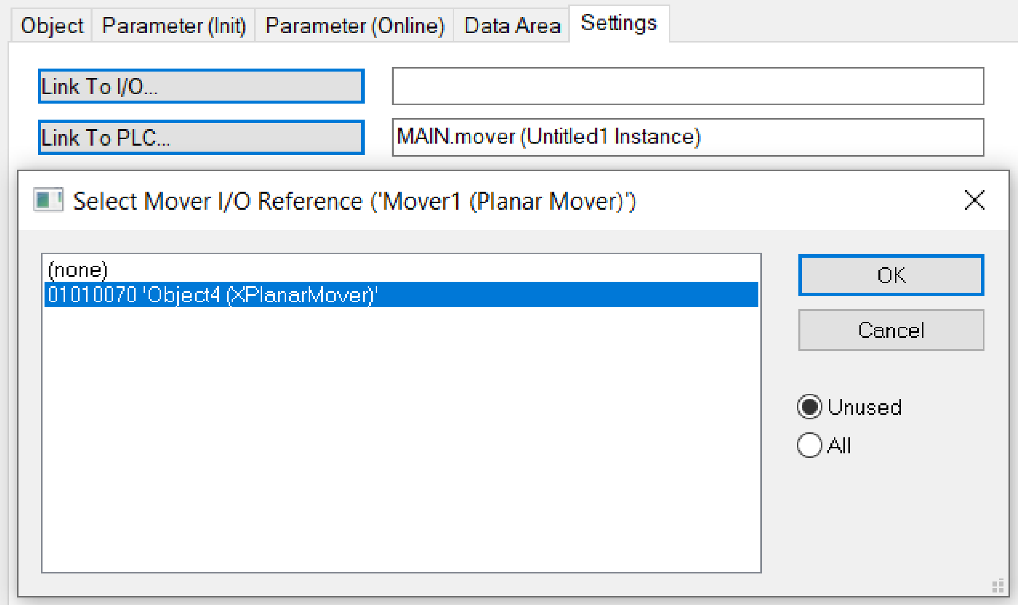
1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



⇨ In addition, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To I/O...** button on the **Settings** tab.



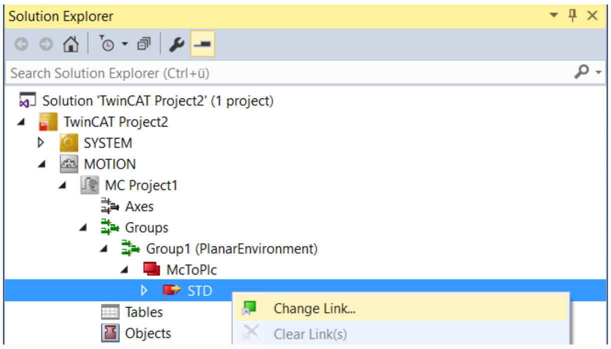
MAIN.mover (Untitled1 Instance)

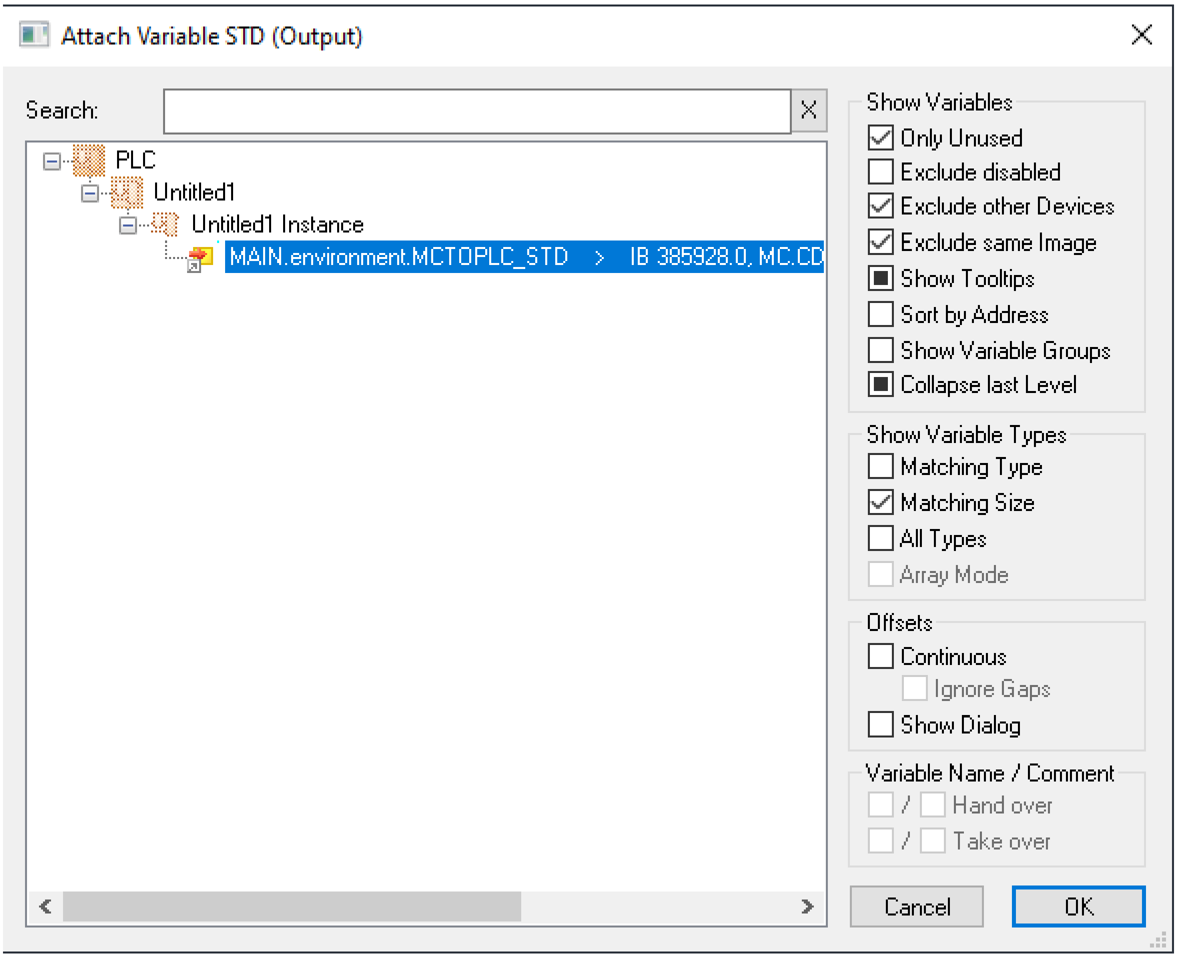
Select Mover I/O Reference ('Mover1 (Planar Mover)')

(none)
01010070 'Object4 (XPlanarMover)'

● Unused
○ All

⇨ The Planar environment can then be linked in the "MC Project".





Attach Variable STD (Output)

Search:

- PLC
  - Untitled1
    - Untitled1 Instance
      - MAIN.environment.MCTOPLC_STD > IB 385928.0, MC.CD

Show Variables
- ☑ Only Unused
- □ Exclude disabled
- ☑ Exclude other Devices
- ☑ Exclude same Image
- ■ Show Tooltips
- □ Sort by Address
- □ Show Variable Groups
- ■ Collapse last Level

Show Variable Types
- □ Matching Type
- ☑ Matching Size
- □ All Types
- □ Array Mode

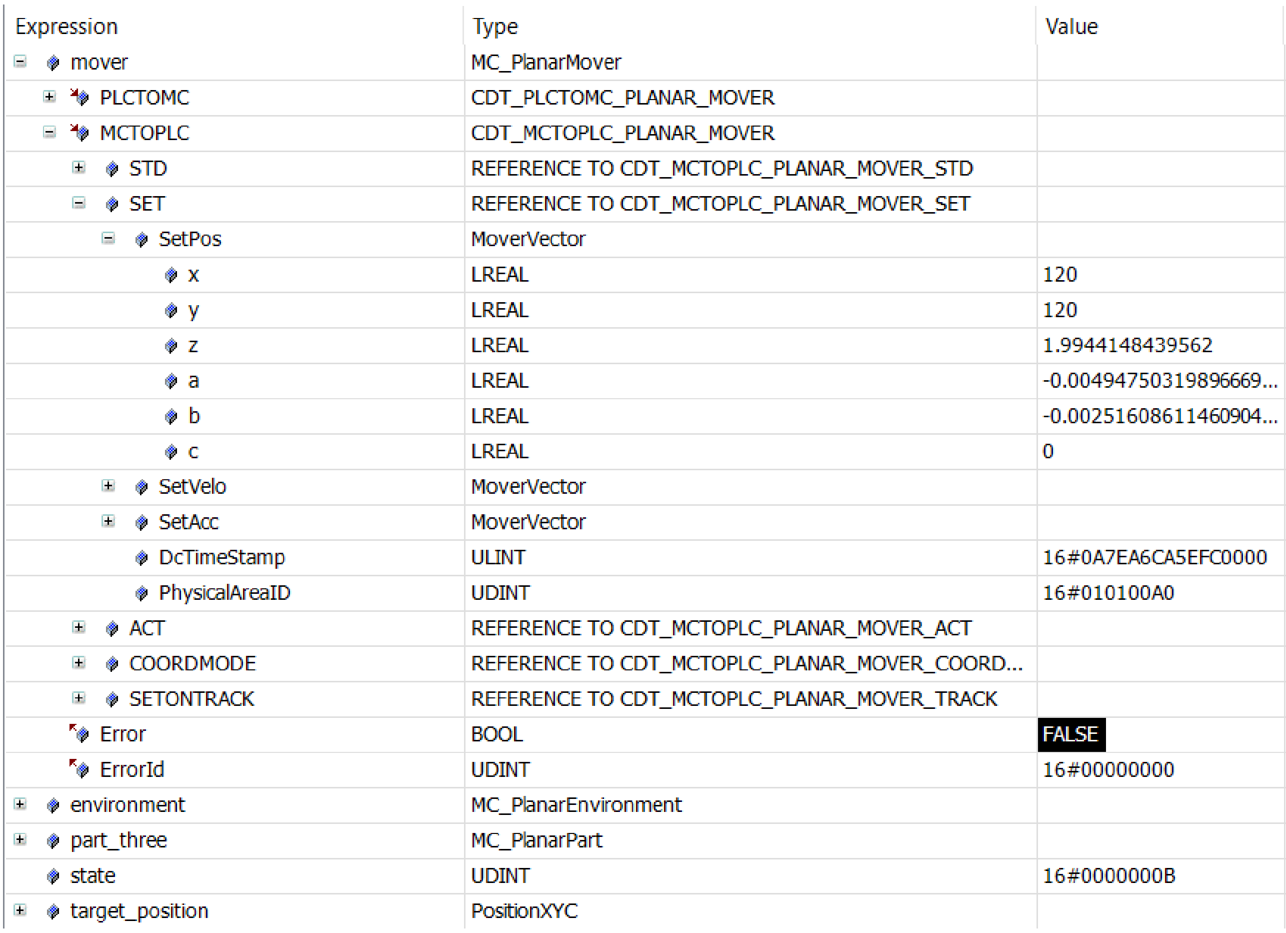
Offsets
- □ Continuous
- □ Ignore Gaps
- □ Show Dialog

Variable Name / Comment
- □ / □ Hand over
- □ / □ Take over

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state = 11, shown in hexadecimal in the figure below as B), the mover is in the desired position. The position is specified in coordinate system two (object Id 16#010100A0). The mover has changed the coordinate system together with part 3, or has moved to a different level with the elevator. Overall, the example clearly shows that the Planar-Part PLC objects are only lightweight environment wrappers that have to be initialized with the environment and send their commands via the environment.



Expression	Type	Value
⊟ mover	MC_PlanarMover	
⊞ PLCTOMC	CDT_PLCTOMC_PLANAR_MOVER	
⊟ MCTOPLC	CDT_MCTOPLC_PLANAR_MOVER	
⊞ STD	REFERENCE TO CDT_MCTOPLC_PLANAR_MOVER_STD	
⊟ SET	REFERENCE TO CDT_MCTOPLC_PLANAR_MOVER_SET	
⊟ SetPos	MoverVector	
x	LREAL	120
y	LREAL	120
z	LREAL	1.9944148439562
a	LREAL	-0.00494750319896669...
b	LREAL	-0.00251608611460904...
c	LREAL	0
⊞ SetVelo	MoverVector	
⊞ SetAcc	MoverVector	
DcTimeStamp	ULINT	16#0A7EA6CA5EFC0000
PhysicalAreaID	UDINT	16#010100A0
⊞ ACT	REFERENCE TO CDT_MCTOPLC_PLANAR_MOVER_ACT	
⊞ COORDMODE	REFERENCE TO CDT_MCTOPLC_PLANAR_MOVER_COORD...	
⊞ SETONTRACK	REFERENCE TO CDT_MCTOPLC_PLANAR_MOVER_TRACK	
Error	BOOL	FALSE
ErrorId	UDINT	16#00000000
⊞ environment	MC_PlanarEnvironment	
⊞ part_three	MC_PlanarPart	
state	UDINT	16#0000000B
⊞ target_position	PositionXYC	

6.7 Planar Feedback

The MC Planar Feedback [⏵ 137] is a PLC software object that bundles all the status information for a command that is given by the user to a mover, track, group or other Planar component.

This ranges from the sending of the command by the user to the processing of the command by the components and from the subsequent acceptance (or possibly rejection) to the execution and termination of the command. The user can track all of this using a feedback object if he so desires.

To do this, he must transfer a feedback object in the PLC as the first argument when the command method is called. Subsequently, whenever the user triggers the feedback object (or calls its update method), he can retrieve the current command state.

In order for a Planar Feedback to be used, it must be declared in the PLC. The Planar Feedback has no fixed equivalent in a TCOM object on the Motion Control side. From there, it receives the information directly from the corresponding TCOM object (e.g. Planar mover), which executes the corresponding command. Therefore, feedback does not need to be created, parameterized or linked separately in the MC project.

6.7.1 Example "Creating a Planar mover and Planar Feedback"

Using this short guide you will create a TwinCAT project that contains a Planar mover and a Planar Feedback.

Creating a Planar mover

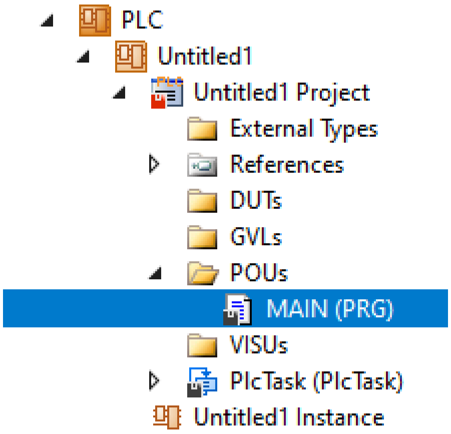
✓ See Configuration [⏵ 18].

1. Create a Planar mover.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a PLC

✓ See preliminary steps under Creating a PLC [▶ 21].

1. Create a mover ("MC_PlanarMover") and a Planar Feedback ("MC_PlanarFeedback") via MAIN as follows.



⇨ These represent the mover and the Planar Feedback in the MC Configuration.

```
PROGRAM MAIN
VAR
    mover : MC_PlanarMover;
    feedback : MC_PlanarFeedback;
    state : UDINT;
    target_position : PositionXYC;
END_VAR
```

In this simple example you have created a state variable for a state machine and a target position for a travel command of the Mover. A feedback is also declared in order to monitor the command process, with which a sequence can subsequently be programmed in the MAIN:

```
CASE state OF
  0:
    mover.Enable(feedback);
    state := 1;
  1:
    IF feedback.Done THEN
      state := 2;
    END_IF
  2:
    target_position.SetValuesXY(1000, 1000);
    mover.MoveToPosition(feedback, target_position, 0, 0);
    state := 3;
END_CASE
```

This program code activates the mover and moves it to position x = 1000 and y = 1000.

Note that the state machine will only be advanced when the feedback signals the successful termination of the command via its "Done" flag.

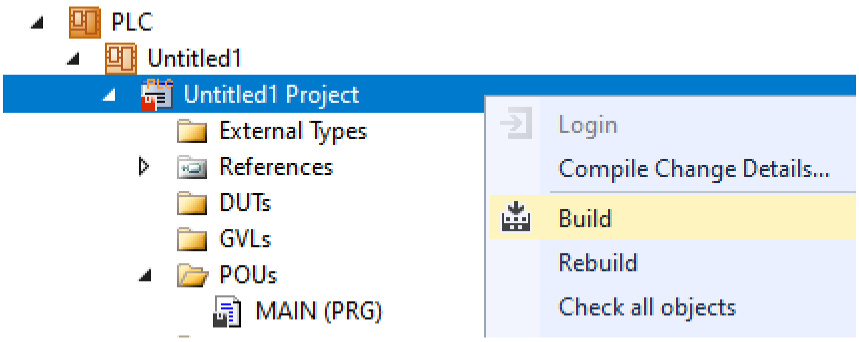
Sending the command

2. To issue the command and monitor the feedback, you must call the mover and feedback cyclically with their update methods after the END_CASE:

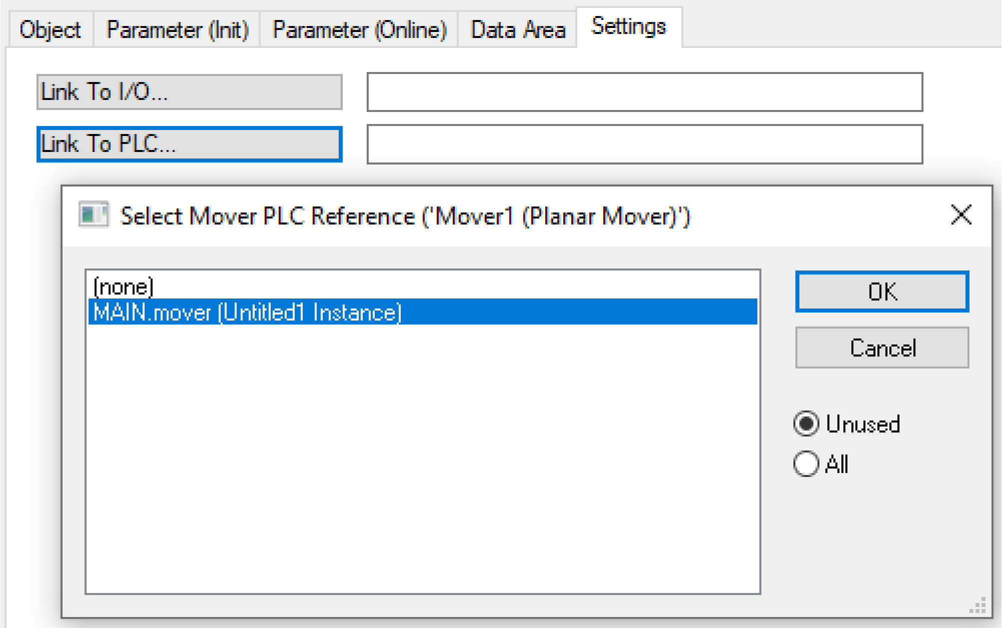
```
mover.Update();
feedback.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



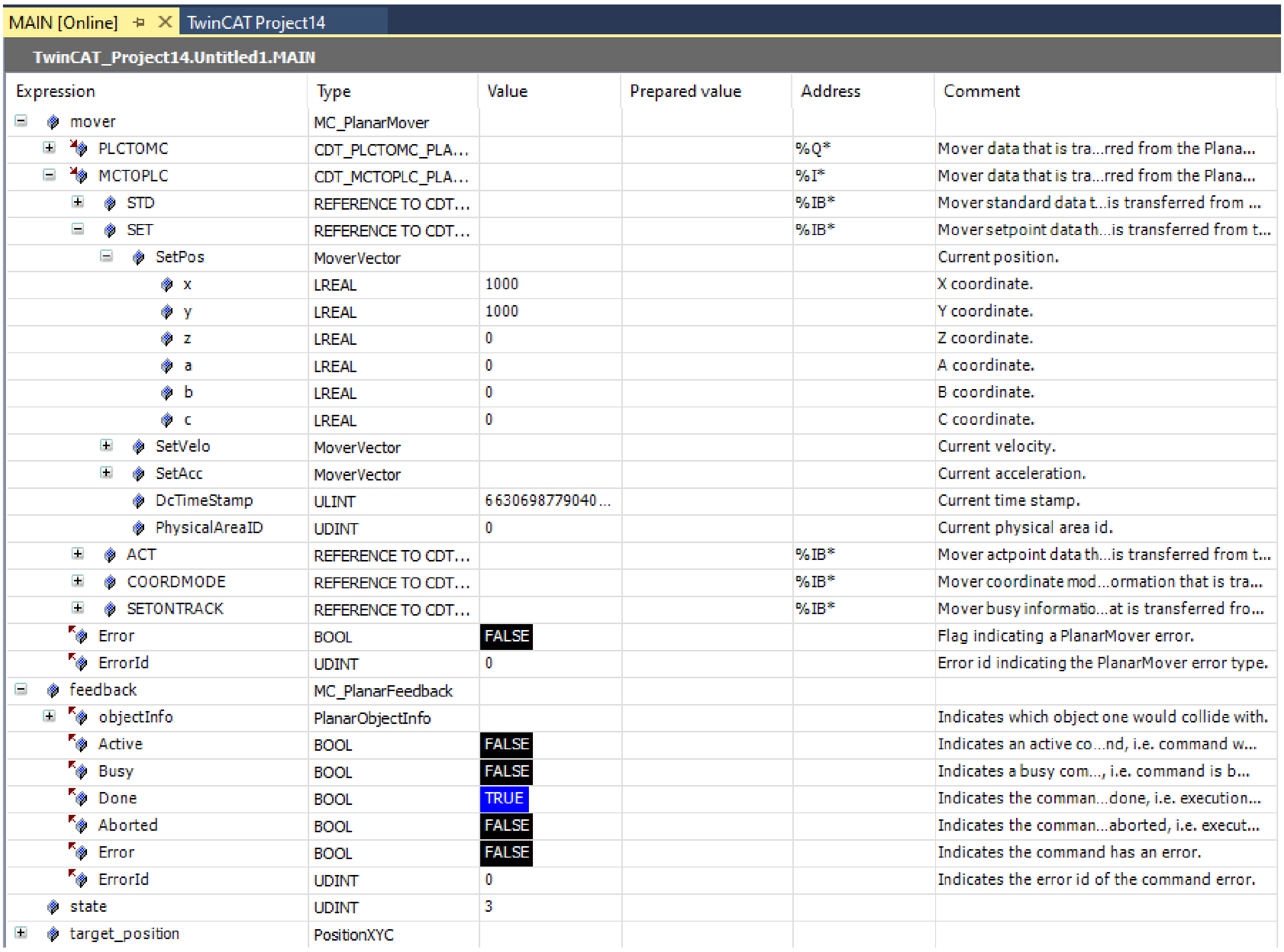
⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.



Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

The mover is in the desired position at the end of the state machine (state = 3) and the feedback signals the termination of the command with the "Done" flag.



Expression	Type	Value	Prepared value	Address	Comment
⊟ mover	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	1000			X coordinate.
y	LREAL	1000			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6630698779040...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
⊟ feedback	MC_PlanarFeedback				
⊞ objectInfo	PlanarObjectInfo				Indicates which object one would collide with.
Active	BOOL	FALSE			Indicates an active co...nd, i.e. command w...
Busy	BOOL	FALSE			Indicates a busy com..., i.e. command is b...
Done	BOOL	TRUE			Indicates the comman...done, i.e. execution...
Aborted	BOOL	FALSE			Indicates the comman...aborted, i.e. execut...
Error	BOOL	FALSE			Indicates the command has an error.
ErrorId	UDINT	0			Indicates the error id of the command error.
state	UDINT	3			
⊞ target_position	PositionXYC				

6.7.2 Example "Planar motion components: averting collision"

Using this brief guide you will create a TwinCAT project that contains a Planar mover whose travel command is rejected due to a collision with the Planar environment.

Creating a Planar mover

✓ See Configuration [▸ 18].

1. Create a Planar mover.
2. Put "Parameter (Init)" into simulation mode (`TRUE`). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.

Creating a Planar environment

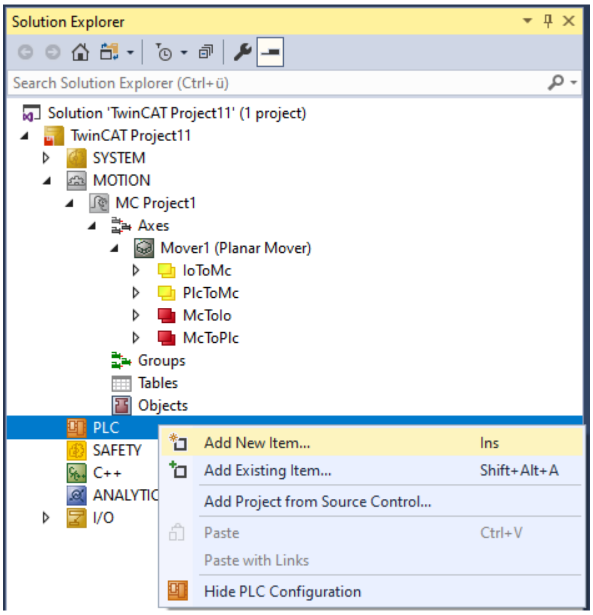
3. Create a Planar environment, see Configuration [▸ 96].

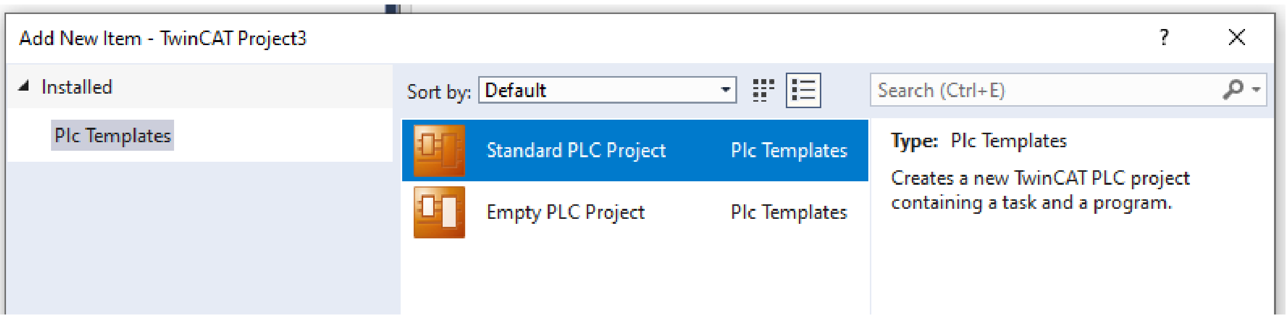
Creating a Planar group

4. Create a Planar group, see Configuration [▸ 89].

Creating a PLC

✓ In order to create the geometry of the environment and control the mover, a PLC must be created from which the user can send commands to both.

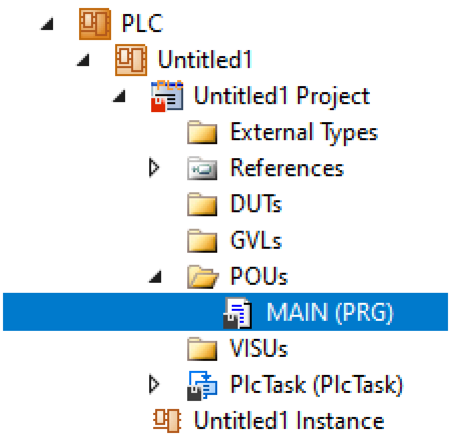



Add New Item - TwinCAT Project3
?
Installed
Plc Templates
Sort by: Default
Search (Ctrl+E)
Standard PLC Project Plc Templates
Empty PLC Project Plc Templates

Type: Plc Templates

Creates a new TwinCAT PLC project containing a task and a program.

5. Add the libraries Tc3_Physics and Tc3_Mc3PlanarMotion to the PLC project, see Inserting libraries [▶ 125].
6. Create an MC_PlanarMover [▶ 143] and an MC_PlanarEnvironment [▶ 134] via **MAIN**.



⇨ These represent the mover and the environment in the MC Configuration.

```
PROGRAM MAIN
VAR
    mover           : MC_PlanarMover;
    environment     : MC_PlanarEnvironment;
    group           : MC_PlanarGroup;
    feedback        : MC_PlanarFeedback;
    state           : UDINT;
    target_position : PositionXYC;
END_VAR
```

In this example you have created a state variable for a simple state machine and a target position for a travel command of the mover, with which a sequence can subsequently be programmed in the MAIN:

```
CASE state OF
  0:
    environment.AddStator(0,-120.0,-120.0);
    environment.CreateBoundary(0);
    state := 1;
  1:
    mover.Enable(0);
    group.Enable(0);
    state := 2;
  2:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled AND
    group.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 3;
    ENDIF
  3:
    mover.AddToGroup(0,group);
    environment.AddToGroup(0,group);
    state := 4;
  4:
    IF mover.MCTOPLC.STD.GroupOID > 0 AND
    environment.MCTOPLC_STD. GroupOID > 0 THEN
      state := 5;
    ENDIF
  5:
    target_position.SetValuesXY(100, 100);
    mover.MoveToPosition(feedback, target_position, 0, 0);
    state := 6;
END_CASE
```

This program code activates the mover and creates an environment from a tile on which the mover is located. An attempt is then made to move the mover to the position x = 100 and y = 100.

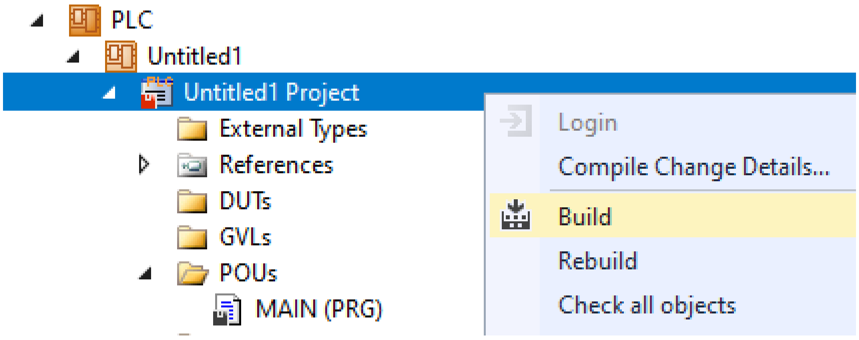
Sending the command

7. In order to issue the command and monitor the feedback, you must call the objects cyclically with their update methods after the END_CASE:

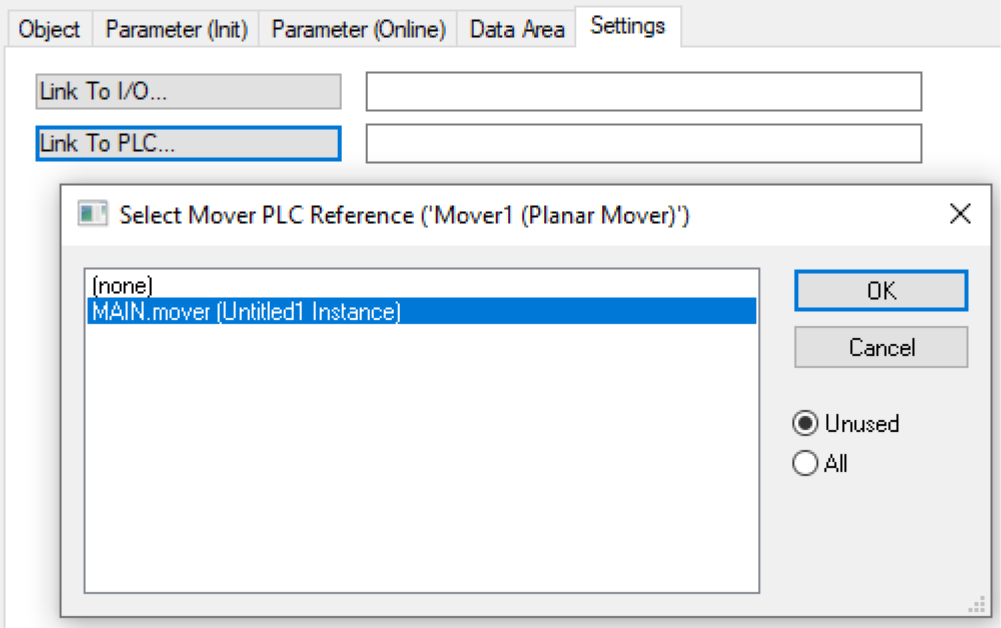
```
mover.Update();
environment.Update();
group.Update();
feedback.Update();
```

When creating the PLC, a symbol of the "PLC Mover" is created, which can then be linked to the mover instance in the MC project.

1. To build, use the path **PLC > Untitled1 > Untitled1 Project > Build.**



⇨ Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the **Link To PLC...** button on the **Settings** tab.

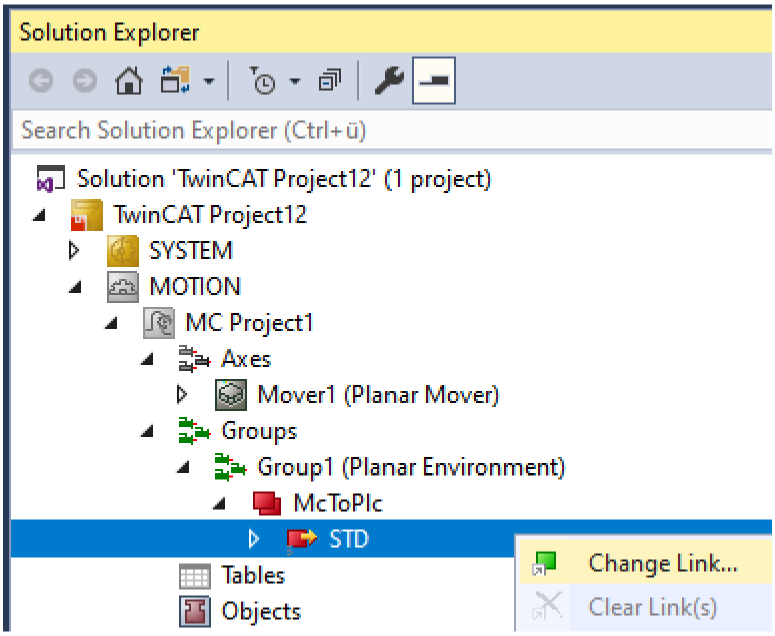


Select Mover PLC Reference ('Mover1 (Planar Mover)')

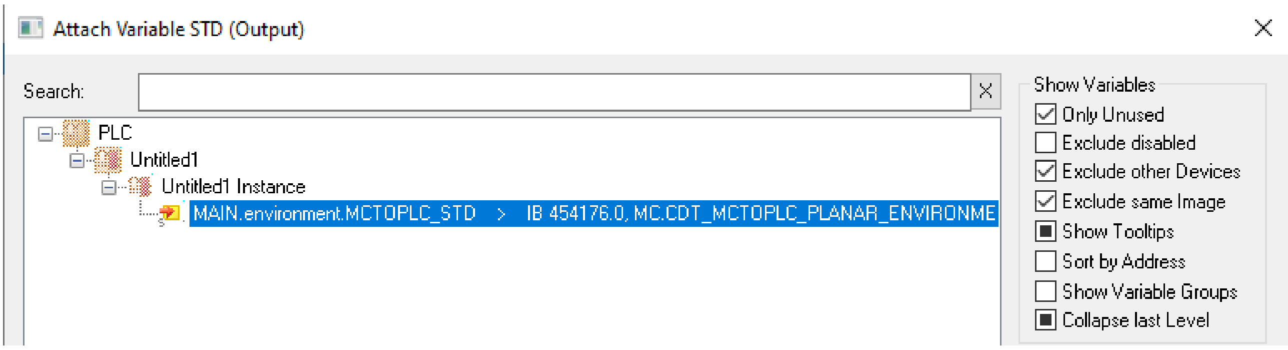
(none)
MAIN.mover (Untitled1 Instance)

● Unused
○ All

⇨ Subsequently, the Planar environment can be linked via the following dialog boxes in the "MC Project".



Solution 'TwinCAT Project12' (1 project)
- TwinCAT Project12
  - SYSTEM
  - MOTION
    - MC Project1
      - Axes
        - Mover1 (Planar Mover)
      - Groups
        - Group1 (Planar Environment)
          - McToPlc
            - STD
      - Tables
      - Objects



- PLC
  - Untitled1
    - Untitled1 Instance
      - MAIN.environment.MCTOPLC_STD > IB 454176.0, MC.CDT_MCTOPLC_PLANAR_ENVIRONME

Show Variables
☑ Only Unused
☐ Exclude disabled
☑ Exclude other Devices
☑ Exclude same Image
■ Show Tooltips
☐ Sort by Address
☐ Show Variable Groups
■ Collapse last Level

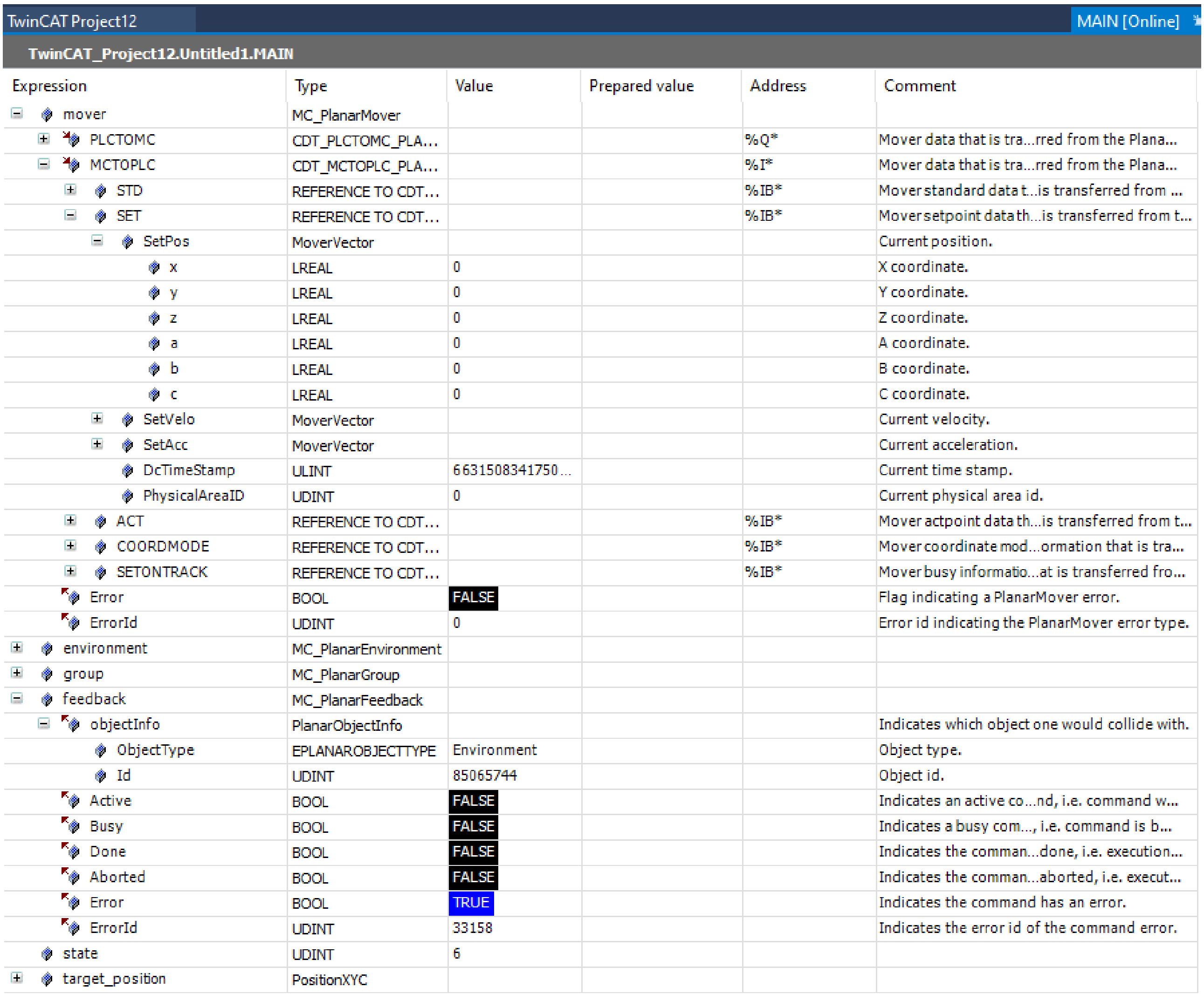
⇨ The group is linked in the same way.

Activating and starting the project

1. Activate the configuration via the button in the menu bar .

2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

At the end of the state machine (state=6), the mover is in the desired position. The mover did not move because the command was rejected. The feedback shows a collision error and the environment is specified as the collision partner in the ObjectInfo.



TwinCAT Project12 MAIN [Online]

TwinCAT_Project12.Untitled1.MAIN

Expression	Type	Value	Prepared value	Address	Comment
⊟ mover	MC_PlanarMover				
⊞ PLCTOMC	CDT_PLCTOMC_PLA...			%Q*	Mover data that is tra...rred from the Plana...
⊟ MCTOPLC	CDT_MCTOPLC_PLA...			%I*	Mover data that is tra...rred from the Plana...
⊞ STD	REFERENCE TO CDT...			%IB*	Mover standard data t...is transferred from ...
⊟ SET	REFERENCE TO CDT...			%IB*	Mover setpoint data th...is transferred from t...
⊟ SetPos	MoverVector				Current position.
x	LREAL	0			X coordinate.
y	LREAL	0			Y coordinate.
z	LREAL	0			Z coordinate.
a	LREAL	0			A coordinate.
b	LREAL	0			B coordinate.
c	LREAL	0			C coordinate.
⊞ SetVelo	MoverVector				Current velocity.
⊞ SetAcc	MoverVector				Current acceleration.
DcTimeStamp	ULINT	6631508341750...			Current time stamp.
PhysicalAreaID	UDINT	0			Current physical area id.
⊞ ACT	REFERENCE TO CDT...			%IB*	Mover actpoint data th...is transferred from t...
⊞ COORDMODE	REFERENCE TO CDT...			%IB*	Mover coordinate mod...ormation that is tra...
⊞ SETONTRACK	REFERENCE TO CDT...			%IB*	Mover busy informatio...at is transferred fro...
Error	BOOL	FALSE			Flag indicating a PlanarMover error.
ErrorId	UDINT	0			Error id indicating the PlanarMover error type.
⊞ environment	MC_PlanarEnvironment				
⊞ group	MC_PlanarGroup				
⊟ feedback	MC_PlanarFeedback				
⊟ objectInfo	PlanarObjectInfo				Indicates which object one would collide with.
ObjectType	EPLANAROBJECTTYPE	Environment			Object type.
Id	UDINT	85065744			Object id.
Active	BOOL	FALSE			Indicates an active co...nd, i.e. command w...
Busy	BOOL	FALSE			Indicates a busy com..., i.e. command is b...
Done	BOOL	FALSE			Indicates the comman...done, i.e. execution...
Aborted	BOOL	FALSE			Indicates the comman...aborted, i.e. execut...
Error	BOOL	TRUE			Indicates the command has an error.
ErrorId	UDINT	33158			Indicates the error id of the command error.
state	UDINT	6			
⊞ target_position	PositionXYC				

6.7.3 Specialized feedback types

In addition to the general MC_PlanarFeedback type, which is accepted by most commands, certain commands may require a specialized feedback type. Planar Feedback [▶ 112] that apply to the general feedback also apply to these types.

Specialized feedbacks can have a subset of the outputs of the general feedback, depending on their type. This includes information about whether a command is active or whether it caused an error, etc. In addition, specialized feedback types may have other outputs or functions that correspond to their scope of application.

MC_PlanarFeedbackGearInPosOnTrack

This feedback type is accepted by a Example "Synchronizing a Planar mover on a track with one axis" [▶ 65]. It has an additional output `inSync`, which indicates whether the executing mover is synchronous with the master axis.

MC_PlanarFeedbackGearInPosOnTrackWithMasterMover

This feedback type is accepted by a Example: "Synchronizing a Planar mover on a track with another Planar mover" [▶ 72]. It has an additional output inSync, which indicates whether the executing mover is synchronous with the Master Planar Mover.

6.8 Planar TrackTrail

The MC_PlanarTrackTrail [▶ 171] is an object that defines a path of contiguous Planar tracks in a network. In contrast to the individual Planar tracks from which the Planar track trail is built, the Planar track trail has no fixed equivalent in a TCOM object on the MC side, but is declared solely in the PLC, similar to a Planar feedback [▶ 112].

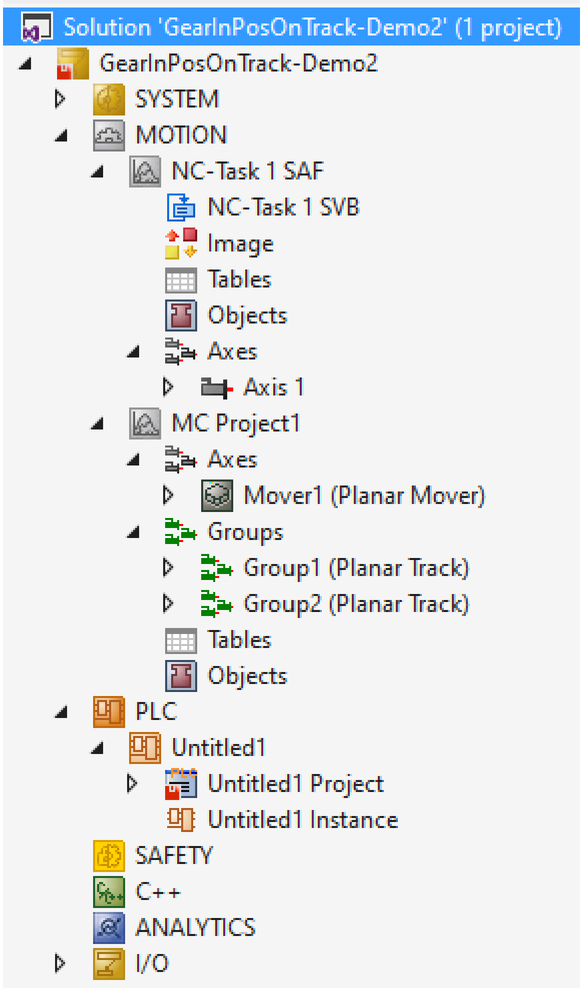
A Planar track trail can be used to define a path of Planar tracks via which a synchronization movement of a Slave Planar Mover with a master axis [▶ 65] or with a Master Planar Mover [▶ 72] should take place (if this path consists of more than the current Planar track of the Slave Planar Mover).

The Planar-TrackTrail offers methods for adding a Planar track to its end and for emptying its configuration. These methods only modify the Planar-TrackTrail and leave the underlying Planar tracks and the network untouched.

When adding a Planar track, make sure that it connects to the end of the current last Planar track in the Planar-TrackTrail. It is also impossible to add a Planar track more than once.

6.8.1 Example "Synchronization movement over two Planar tracks"

This example is an extension of the example Example "Synchronizing a Planar mover on a track with one axis" [▶ 65], in which the synchronization movement of the Planar mover takes place over two Planar tracks. The above example is modified so that *two* Planar tracks are created in the MC Configuration. The Solution Explorer then has the following entries:



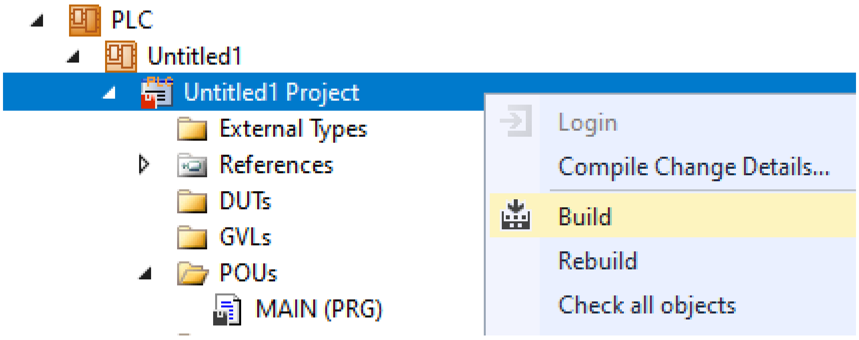
```
Solution 'GearInPosOnTrack-Demo2' (1 project)
◢ GearInPosOnTrack-Demo2
  ▷ SYSTEM
  ◢ MOTION
    ◢ NC-Task 1 SAF
        NC-Task 1 SVB
        Image
        Tables
        Objects
      ◢ Axes
        ▷ Axis 1
    ◢ MC Project1
      ◢ Axes
        ▷ Mover1 (Planar Mover)
      ◢ Groups
        ▷ Group1 (Planar Track)
        ▷ Group2 (Planar Track)
        Tables
        Objects
  ◢ PLC
    ◢ Untitled1
      ▷ Untitled1 Project
        Untitled1 Instance
    SAFETY
    C++
    ANALYTICS
  ▷ I/O
```

Customizing the PLC project

1. Add the libraries Tc3_Mc3PlanarMotion, Tc3_Physics and Tc2_MC2 to the PLC project; see Inserting libraries [▶ 125].
2. Declare the following variables in MAIN:

```
PROGRAM MAIN
VAR
    mover       : MC_PlanarMover;
    track1      : MC_PlanarTrack;
    track2      : MC_PlanarTrack;
    trail       : MC_PlanarTrackTrail;
    axis        : AXIS_REF;
    power_axis  : MC_Power;
    move_axis   : MC_MoveAbsolute;
    state       : UDINT;
    pos1, pos2  : PositionXYC;
END_VAR
```

3. Configure the PLC to create symbols of the "PLC mover", the "PLC tracks" and the "PLC axis".



4. Link the Planar movers, Planar tracks (see example "Example: "Creating and moving Planar movers" [▶ 22]") and the axis (see example "Example "Synchronizing a Planar mover on a track with one axis" [▶ 65]").

All the steps so far, except for doubling the number of Planar tracks and the slightly modified code, are identical to those in the example Example "Synchronizing a Planar mover on a track with one axis" [▶ 65].

Programming state machines

The next step is to modify the program code so that the Planar TrackTrail is passed to the `GearInPosOnTrack` command. Before that the Planar TrackTrail is populated with both Planar tracks, which in this example form a simple Example "Connecting Planar tracks to a network" [▶ 75], consisting of an L-configuration with a loop piece:

```
CASE state OF
  0:
    pos1.SetValuesXYC(100, 100, 0);
    pos2.SetValuesXYC(400, 100, 0);
    track1.AppendLine(0, pos1, pos2);
    track1.Enable(0);
    state := state + 1;
  1:
    IF track1.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  2:
    track2.StartFromTrack(0, track1);
    state := state + 1;
  3:
    pos1.SetValuesXYC(500, 100, 0);
    pos2.SetValuesXYC(860, 100, 0);
    track2.AppendLine(0, pos1, pos2);
    track2.Enable(0);
    state := state + 1;
  4:
    IF track2.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  5:
    mover.Enable(0);
    state := state + 1;
  6:
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
    END_IF
  7:
    mover.JoinTrack(0, track1, 0, 0);
    state := state + 1;
  8:
    IF mover.MCTOPLC.STD.CommandMode =
    MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
    END_IF
  9:
    power_axis(Axis := axis,
    Enable := TRUE,
    Enable_Positive := TRUE);
    IF power_axis.Status THEN
```

```
      move_axis(Axis := axis, Execute := FALSE);
      state := state + 1;
    END_IF
  10:
    move_axis(Axis := axis,
              Position := 700,
              Velocity := 30,
              Acceleration := 100,
              Deceleration := 100,
              Jerk := 100,
              Execute := TRUE);
    state := state + 1;
  11:
    trail.AddTrack(track1);
    trail.AddTrack(track2);
    mover.GearInPosOnTrack(0, axis.DriveAddress.TcAxisObjectId, trail, 100, 100, track1, 0, 0);
    state := state + 1;
END_CASE

mover.Update();
track1.Update();
track2.Update();
power_axis(Axis := axis);
move_axis(Axis := axis);
axis.ReadStatus();
```

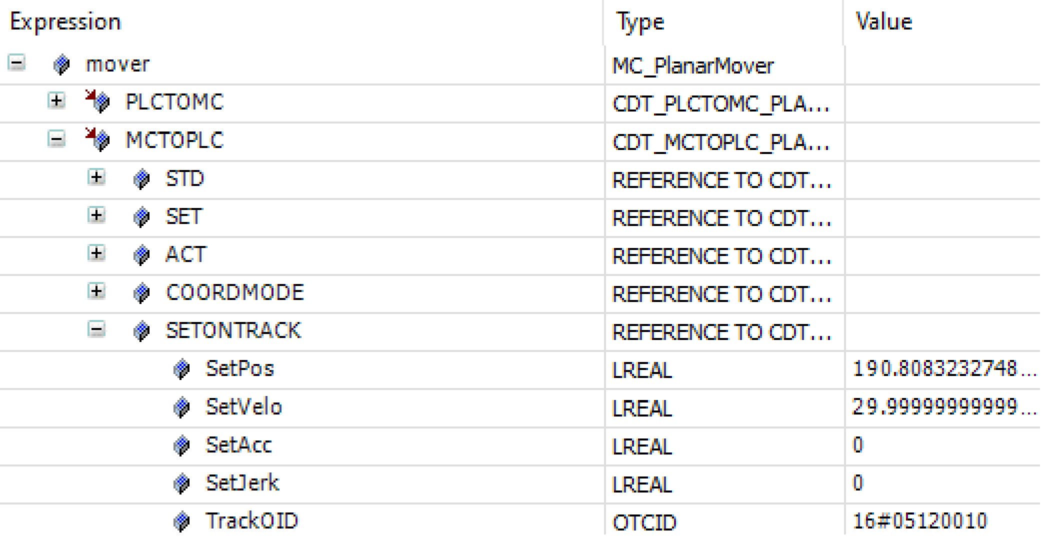
The two Planar tracks are added to the Planar TrackTrail in State 11. The order is crucial here, since `track2` follows `track1` and not vice versa. The Planar TrackTrail is passed as the third argument to the `GearInPosOnTrack` command.

Activating and starting the project

1. Activate the configuration via the button in the menu bar .
2. Set the TwinCAT system to the "Run" state via the button.
3. Log in the PLC via the button in the menu bar .
4. Start the PLC via the Play button in the menu bar.

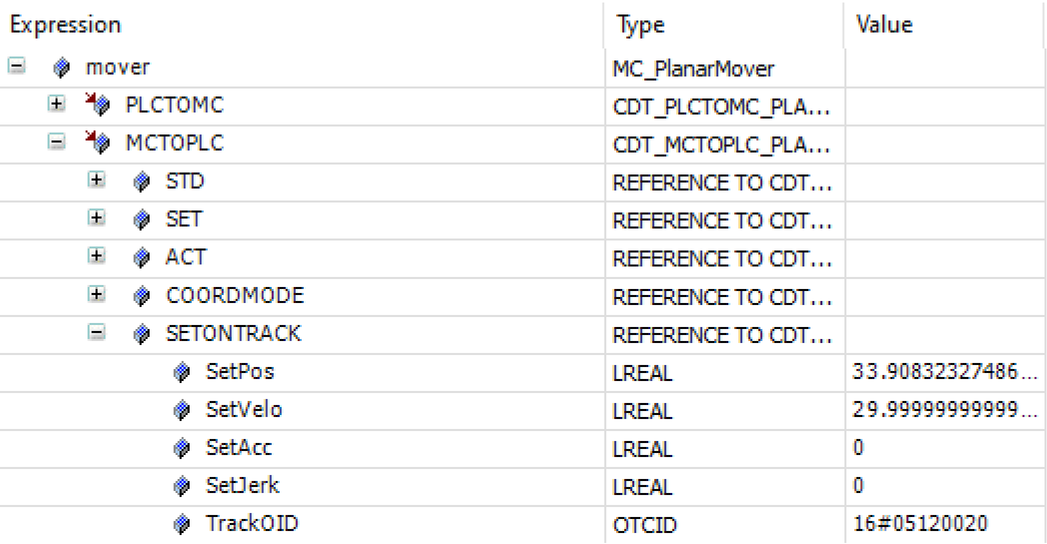
Observe the process in the online view

5. Note in the online view how the Planar mover initially moves along the first Planar track towards its end:



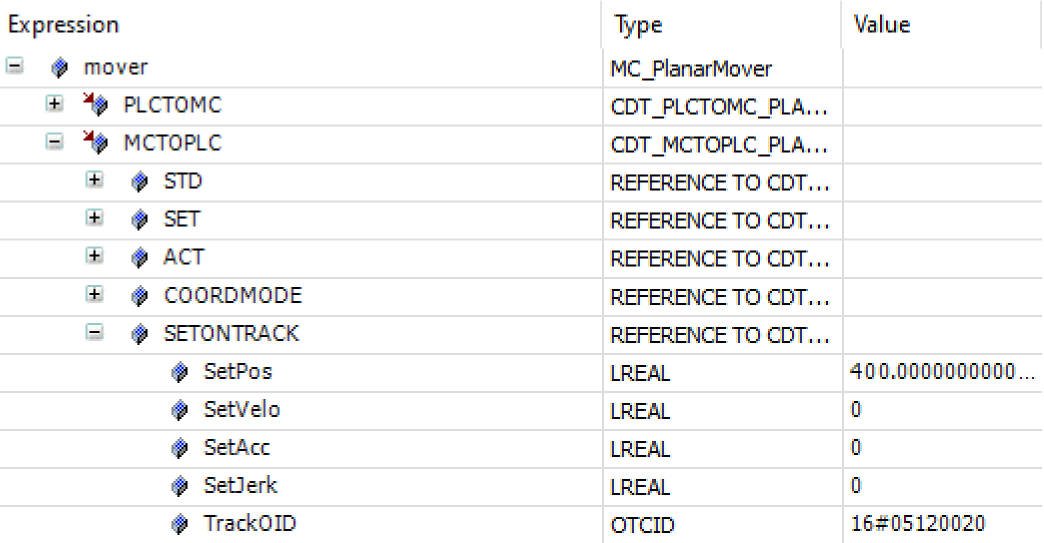
Expression	Type	Value
mover	MC_PlanarMover	
PLCTOMC	CDT_PLCTOMC_PLA...	
MCTOPLC	CDT_MCTOPLC_PLA...	
STD	REFERENCE TO CDT...	
SET	REFERENCE TO CDT...	
ACT	REFERENCE TO CDT...	
COORDMODE	REFERENCE TO CDT...	
SETONTRACK	REFERENCE TO CDT...	
SetPos	LREAL	190.8083232748...
SetVelo	LREAL	29.99999999999...
SetAcc	LREAL	0
SetJerk	LREAL	0
TrackOID	OTCID	16#05120010

6. You will then see it switch to the subsequent Planar track (note the TrackOIDs):



Expression	Type	Value
⊟ mover	MC_PlanarMover	
⊞ PLCTOMC	CDT_PLCTOMC_PLA...	
⊟ MCTOPLC	CDT_MCTOPLC_PLA...	
⊞ STD	REFERENCE TO CDT...	
⊞ SET	REFERENCE TO CDT...	
⊞ ACT	REFERENCE TO CDT...	
⊞ COORDMODE	REFERENCE TO CDT...	
⊟ SETONTRACK	REFERENCE TO CDT...	
SetPos	LREAL	33.90832327486...
SetVelo	LREAL	29.99999999999...
SetAcc	LREAL	0
SetJerk	LREAL	0
TrackOID	OTCID	16#05120020

7. Finally, you can see how it comes to a standstill on the second Planar track:



Expression	Type	Value
⊟ mover	MC_PlanarMover	
⊞ PLCTOMC	CDT_PLCTOMC_PLA...	
⊟ MCTOPLC	CDT_MCTOPLC_PLA...	
⊞ STD	REFERENCE TO CDT...	
⊞ SET	REFERENCE TO CDT...	
⊞ ACT	REFERENCE TO CDT...	
⊞ COORDMODE	REFERENCE TO CDT...	
⊟ SETONTRACK	REFERENCE TO CDT...	
SetPos	LREAL	400.0000000000...
SetVelo	LREAL	0
SetAcc	LREAL	0
SetJerk	LREAL	0
TrackOID	OTCID	16#05120020

⇨ Also in this example, the Planar mover will abort its synchronization movement if the behavior of the master axis should require it to pass over the end of the second Planar track (e.g. by making the target position of the master axis greater than the sum of the lengths of the two Planar tracks). In this case the Planar mover comes to a halt at the end of the second track, loses its potential synchronization status and reports an error.

If another Planar track is added to the end of the *first* track so that a switch is created at its end, the Planar mover "knows" unambiguously through the Planar TrackTrail to which of the two Planar tracks it should turn and thus continue its synchronization movement (after all, the master axis produces its setpoints independently of Planar tracks). In this way, a Planar TrackTrail can be used to perform a synchronization movement through track networks of any complexity on a unique path of any length.

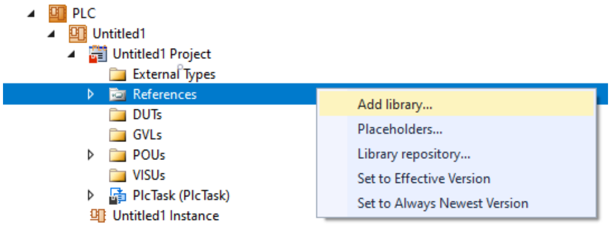
Since the Planar TrackTrail is a pure PLC object that does not communicate via TCOM but only acts as a container, no cyclic update, as for example for the Planar mover, the Planar tracks or Planar Feedback [▶ 112] (which are not used in this example), is necessary, and a corresponding method is not available.

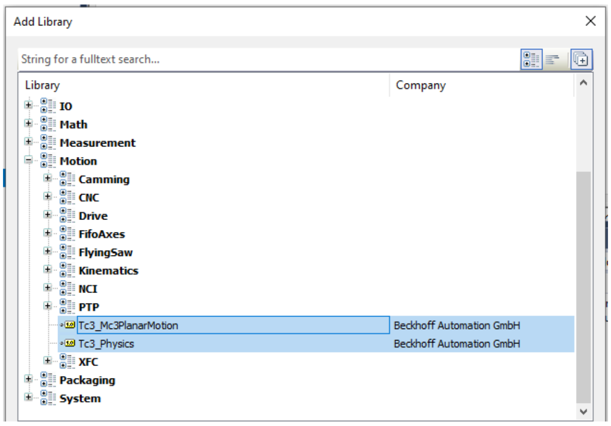
7 PLC Libraries

7.1 Inserting libraries

✓ The libraries "Tc3_Physics" and "Tc3_Mc3PlanarMotion" must be integrated in order to control XPlanar components.

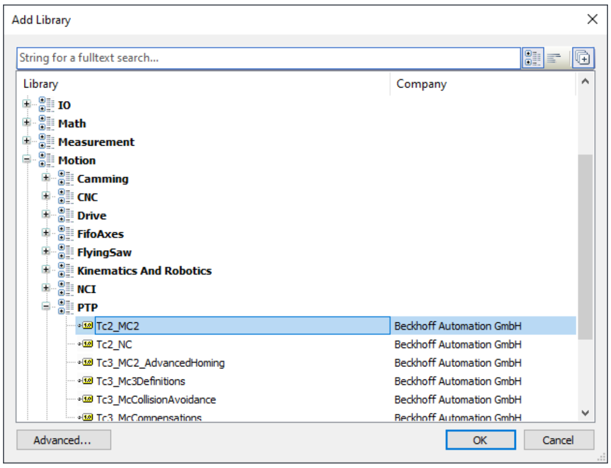
1. Add the desired libraries to your project one after the other via **References > Add library...**





⇨ Once the libraries are integrated, you can program the rest of the process in the PLC.

To control a master axis, the library "Tc2_MC2" must also be included.



7.2 Tc3_Mc3PlanarMotion API

7.2.1 Data Types

7.2.1.1 Enums

7.2.1.1.1 EPlanarObjectType

Identifies a planar object type.

Syntax

Definition:

```
TYPE EPlanarObjectType :
(
    Invalid     := 0,
    None        := 301,
    Mover       := 302,
    Track       := 303,
    Environment := 304
)UINT;
END_TYPE
```

Values

Name	Description
Invalid	Indicates invalid information, e.g. no connection or component not yet ready.
None	No planar object.
Mover	Planar Mover.
Track	Planar Track.
Environment	Planar Environment.

7.2.1.1.2 MC_DIRECTION

Indicates the movement direction of the Planar Mover on a Planar Track.

Syntax

Definition:

```
TYPE MC_DIRECTION :
(
    mcDirectionNonModulo   := 0,
    mcDirectionPositive    := 1,
    mcDirectionShortestWay := 2,
    mcDirectionNegative    := 3
)UINT;
END_TYPE
```

Values

Name	Description
mcDirectionNonModulo	The Planar Mover moves to the absolute value of the target position. Depending on the current position, this may induce forward or backward movement. On looped tracks, multiple passes are possible.
mcDirectionPositive	The Planar Mover moves to the target position in a forward direction. No backward movement is allowed.
mcDirectionShortestWay	The Planar Mover takes the shortest way to the target position. May induce forward or backward movement.
mcDirectionNegative	The Planar Mover moves to the target position in a backward direction. No forward movement is allowed.

In combination with the Tc2_MC2 library it is possible that the data type cannot be resolved uniquely (ambiguous use of name 'MC_Direction'). In this case you have to specify the namespace when using the data type (Tc3_Mc3PlanarMotion.MC_DIRECTION, Tc3_Mc3Definitions.MC_DIRECTION or Tc2_MC2.MC_DIRECTION).

7.2.1.1.3 MC_SYNC_DIRECTIONS

Directions in which a slave is allowed to move during synchronizing phase.

Syntax

Definition:

```
TYPE MC_SYNC_DIRECTIONS :
(
    Positive := 1,
    Negative := 2,
    Both     := 3
)UINT;
END_TYPE
```

Values

Name	Description
Positive	Movement is allowed only in positive direction while synchronizing.
Negative	Movement is allowed only in negative direction while synchronizing.
Both	Movement is allowed in any direction while synchronizing.

7.2.1.2 Structs

7.2.1.2.1 CDT_MCTOPLC_PLANAR_MOVER

Contains the information of the Planar Mover passed from MC to PLC.

Syntax

Definition:

```
TYPE CDT_MCTOPLC_PLANAR_MOVER :
STRUCT
    STD        : Reference To CDT_MCTOPLC_PLANAR_MOVER_STD;
    SET        : Reference To CDT_MCTOPLC_PLANAR_MOVER_SET;
    ACT        : Reference To CDT_MCTOPLC_PLANAR_MOVER_ACT;
    COORDMODE  : Reference To CDT_MCTOPLC_PLANAR_MOVER_COORDMODE;
    SETONTRACK : Reference To CDT_MCTOPLC_PLANAR_MOVER_TRACK;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Description
STD	Reference To CDT_MCTOPLC_PLANAR_MOVER_STD	Mover standard data that is transferred from the Planar Mover to this function block.
SET	Reference To CDT_MCTOPLC_PLANAR_MOVER_SET	Mover setpoint data that is transferred from the Planar Mover to this function block.
ACT	Reference To CDT_MCTOPLC_PLANAR_MOVER_ACT	Mover actpoint data that is transferred from the Planar Mover to this function block.
COORDMODE	Reference To CDT_MCTOPLC_PLANAR_MOVER_COORDMODE	Mover coordinate mode information that is transferred from the Planar Mover to this function block.
SETONTRACK	Reference To CDT_MCTOPLC_PLANAR_MOVER_TRACK	Mover busy information that is transferred from the Planar Mover to this function block.

7.2.1.2.2 CDT_PLCTOMC_PLANAR_MOVER

Contains the information of the Planar Mover passed from PLC to MC.

Syntax

Definition:

```
TYPE CDT_PLCTOMC_PLANAR_MOVER :
STRUCT
    STD : Reference To CDT_PLCTOMC_PLANAR_MOVER_STD;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Description
STD	Reference To CDT_PLCTOMC_PLANAR_MOVER_STD	Mover standard data that is transferred from this function block to the Planar Mover.

7.2.1.2.3 PlanarObjectInfo

Identifies a planar object uniquely by object id and type.

Syntax

Definition:

```
TYPE PlanarObjectInfo :
STRUCT
    ObjectType : EPlanarObjectType;
    Id         : UDINT;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Description
ObjectType	EPlanarObjectType [▶ 125]	Object type.
Id	UDINT	Object id.

7.2.1.2.4 ST_AdoptTrackOrientationOptions

Options for the "AdoptTrackOrientation" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_AdoptTrackOrientationOptions :
STRUCT
    additionalTurns  : UDINT;
    direction        : MC_DIRECTION;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
additionalTurns	UDINT	0	Addition turns to move in modulo movement (positive or negative).
direction	MC_DIRECTION [▶ 126]	MC_DIRECTION.mcDirectionShortestWay	Direction in which the target is approached.

7.2.1.2.5 ST_EndAtTrackAdvancedOptions

Options for the "EndAtTrackAdvanced" command of the Planar Track.

Syntax

Definition:

```
TYPE ST_EndAtTrackAdvancedOptions :
STRUCT
    thisTrackPartPositionIndex       : UDINT;
    otherTrackPartPositionIndex      : UDINT;
    linkOnlyInSpecifiedPartPositions : BOOL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
thisTrackPartPositionIndex	UDINT	0	The index of the position in which the part of this track is for track connection.
otherTrackPartPositionIndex	UDINT	0	The index of the position in which the part of the other track is for track connection.
linkOnlyInSpecifiedPartPositions	BOOL	FALSE	If false the tracks are connected not only in the given positions configuration of their parts but also in all other (geometrically compatible) locations, otherwise only the specified location is connected.

7.2.1.2.6 ST_ExternalSetpointGenerationOptions

Options for the "ExternalSetpointGeneration" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_ExternalSetpointGenerationOptions :
STRUCT
    mode : MC_EXTERNAL_SET_POSITION_MODE;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
mode	MC_EXTERNAL_SET_POSITION_MODE	MC_EXTERNAL_SET_POSITION_MODE.Absolute	Mode can be relative or absolute, relative can be used parallel to all other commands, absolute only alone.

7.2.1.2.7 ST_GearInPosOnTrackOptions

Options for the "GearInPosOnTrack" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_GearInPosOnTrackOptions :
STRUCT
    gap                              : LREAL;
    inSyncToleranceDistance          : LREAL;
    directionSlaveSyncPosition       : MC_DIRECTION;
    moduloToleranceSlaveSyncPosition : LREAL;
    allowedSlaveSyncDirections       : MC_SYNC_DIRECTIONS;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
gap	LREAL	200.0	Minimal distance to next Planar Mover on track.
inSyncToleranceDistance	LREAL	0.0	Tolerance in absolute value of position difference to master axis for inSync flag.
directionSlaveSyncPosition	MC_DIRECTION [▶ 126]	MC_DIRECTION.mcDirectionNonModulo	Direction in which the slave sync position is approached.
moduloToleranceSlaveSyncPosition	LREAL	0.0	Tolerance "window" for slave sync position.
allowedSlaveSyncDirections	MC_SYNC_DIRECTIONS [▶ 126]	MC_SYNC_DIRECTIONS.Positive	Directions in which the slave is allowed to move while in synchronizing phase.

7.2.1.2.8 ST_GearInPosOnTrackWithMasterMoverOptions

Options for the "GearInPosOnTrackWithMasterMover" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_GearInPosOnTrackWithMasterMoverOptions :
STRUCT
    gap                               : LREAL;
    inSyncToleranceDistance           : LREAL;
    directionSlaveSyncPosition        : MC_DIRECTION;
    moduloToleranceSlaveSyncPosition  : LREAL;
    directionMasterSyncPosition       : MC_DIRECTION;
    moduloToleranceMasterSyncPosition : LREAL;
    allowedSlaveSyncDirections        : MC_SYNC_DIRECTIONS;
    followMover                       : BOOL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
gap	LREAL	200.0	Minimal distance to next Planar Mover on track.
inSyncToleranceDistance	LREAL	0.0	Tolerance in absolute value of position difference to master axis for inSync flag.

Name	Type	Default	Description
directionSlaveSyncPosition	MC_DIRECTION [▶ 126]	MC_DIRECTION.mcDirectionNonModulo	Direction in which the slave sync position is approached.
moduloToleranceSlaveSyncPosition	LREAL	0.0	Tolerance "window" for slave sync position.
directionMasterSyncPosition	MC_DIRECTION [▶ 126]	MC_DIRECTION.mcDirectionNonModulo	Direction in which the master sync position is approached.
moduloToleranceMasterSyncPosition	LREAL	0.0	Tolerance "window" for master sync position.
allowedSlaveSyncDirections	MC_SYNC_DIRECTIONS [▶ 126]	MC_SYNC_DIRECTIONS .Positive	Directions in which the slave is allowed to move while in synchronizing phase.
followMover	BOOL	FALSE	If true, the slave PlanarMover will proceed to follow the master PlanarMover after the latter has traversed the masterSyncPosition. In this case the PlanarTrackTrail describes the slave's path towards the masterSyncPosition. If false, the slave moves exclusively on the PlanarTrackTrail specified.

7.2.1.2.9 ST_JoinTrackOptions

Options for the "JoinTrack" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_JoinTrackOptions :
STRUCT
    useOrientation : BOOL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
useOrientation	BOOL	TRUE	If true, the target orientation is also reached at the end of the movement.

7.2.1.2.10 ST_LeaveTrackOptions

Options for the "LeaveTrack" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_LeaveTrackOptions :
STRUCT
    useOrientation : BOOL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
useOrientation	BOOL	TRUE	If true, the target orientation is also reached at the end of the movement.

7.2.1.2.11 ST_MoveCOptions

Options for the "MoveC" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_MoveCOptions :
STRUCT
    additionalTurns : UDINT;
    direction       : MC_DIRECTION;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
additionalTurns	UDINT	0	Addition turns to move in modulo movement (positive or negative).
direction	MC_DIRECTION [▶ 126]	MC_DIRECTION.mcDirectionNonModulo	Direction in which the target is approached.

7.2.1.2.12 ST_MoveOnTrackOptions

Options for the "MoveOnTrack" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_MoveOnTrackOptions :
STRUCT
    gap             : LREAL;
    direction       : MC_DIRECTION;
    additionalTurns : UDINT;
    moduloTolerance : LREAL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
gap	LREAL	200.0	Minimal distance to next Planar Mover on track.
direction	MC_DIRECTION [▶ 126]	MC_DIRECTION.mcDirectionNonModulo	Direction in which the target is approached.
additionalTurns	UDINT	0	Addition turns to move in modulo movement (positive or negative).

Name	Type	Default	Description
moduloTolerance	LREAL	0.0	Tolerance "window" in modulo movement.

7.2.1.2.13 ST_MoveToPositionOptions

Options for the "MoveToPosition" command of the Planar Mover.

Syntax

Definition:

```
TYPE ST_MoveToPositionOptions :
STRUCT
    useOrientation : BOOL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
useOrientation	BOOL	TRUE	If true, the target orientation is also reached at the end of the movement.

7.2.1.2.14 ST_StartFromTrackAdvancedOptions

Options for the "StartFromTrackAdvanced" command of the Planar Track.

Syntax

Definition:

```
TYPE ST_StartFromTrackAdvancedOptions :
STRUCT
    thisTrackPartPositionIndex       : UDINT;
    otherTrackPartPositionIndex      : UDINT;
    linkOnlyInSpecifiedPartPositions : BOOL;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
thisTrackPartPositionIndex	UDINT	0	The index of the position in which the part of this track is for track connection.
otherTrackPartPositionIndex	UDINT	0	The index of the position in which the part of the other track is for track connection.
linkOnlyInSpecifiedPartPositions	BOOL	FALSE	If false the tracks are connected not only in the given positions configuration of their parts but also in all other (geometrically compatible) locations, otherwise only the specified location is connected.

7.2.2 Function Blocks

7.2.2.1 MC_PlanarEnvironment

A Planar Environment object specifies the environment that Planar Movers can move in. It contains information about the stator objects and boundaries of the movement area.

Do not call the main FB directly. Only use the available methods.

Methods

Name	Description
Clear [▶ 134]	Clears the Planar Environment (stators and boundary).
AddStator [▶ 135]	Adds a stator to the Planar Environment.
CreateBoundary [▶ 135]	Creates a boundary for the Planar Environment based on the previously added stator information or hardware information.
Update [▶ 135]	Updates internal state of the object, must be triggered each cycle.
AddToGroup [▶ 136]	Adds the Planar Environment to the given Planar Group.
RemoveFromGroup [▶ 136]	Removes the Planar Environment from its current Planar Group, i.e. disables collision checks.
GetPlanarObjectInfo [▶ 137]	Returns environment object info (type: environment, id: OID of nc environment).

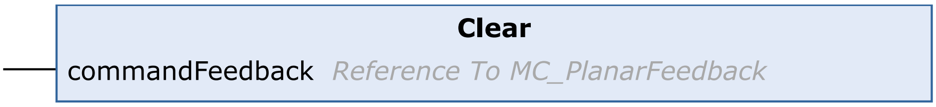
Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.1.1 Clear



Clears the Planar Environment (stators and boundary).

Syntax

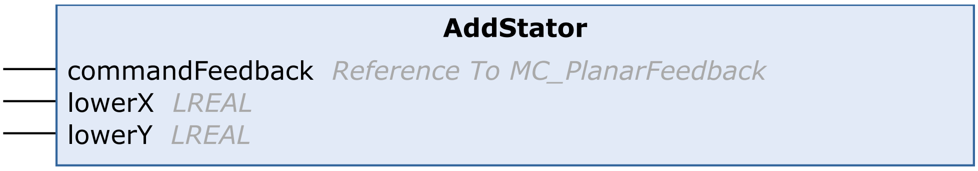
Definition:

```
METHOD Clear
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.1.2 AddStator



Adds a stator to the Planar Environment.

Syntax

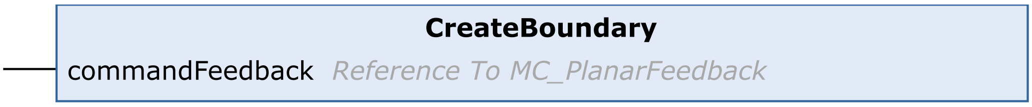
Definition:

```
METHOD AddStator
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    lowerX          : LREAL;
    lowerY          : LREAL;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
lowerX	LREAL	The lower x position of the stator.
lowerY	LREAL	The lower y position of the stator.

7.2.2.1.3 CreateBoundary



Creates a boundary for the Planar Environment based on the previously added stator information or hardware information.

Syntax

Definition:

```
METHOD CreateBoundary
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.1.4 Update

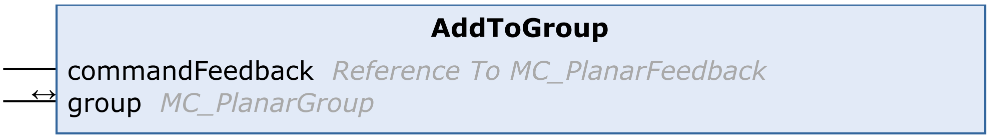
Updates internal state of the object, must be triggered each cycle.

Syntax

Definition:

```
METHOD Update
```

7.2.2.1.5 AddToGroup



Adds the Planar Environment to the given Planar Group.

Syntax

Definition:

```
METHOD AddToGroup
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    group           : MC_PlanarGroup;
END_VAR
```

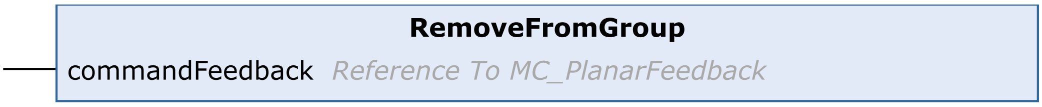
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
group	MC_PlanarGroup [▸ 141]	The Planar Group that the mover joins.

7.2.2.1.6 RemoveFromGroup



Removes the Planar Environment from its current Planar Group, i.e. disables collision checks.

Syntax

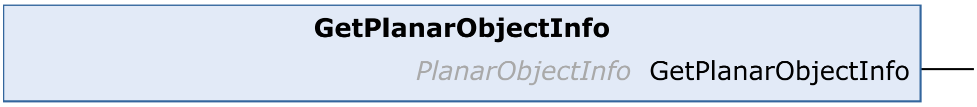
Definition:

```
METHOD RemoveFromGroup
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

7.2.2.1.7 GetPlanarObjectInfo



Returns environment object info (type: environment, id: OID of nc environment).

Syntax

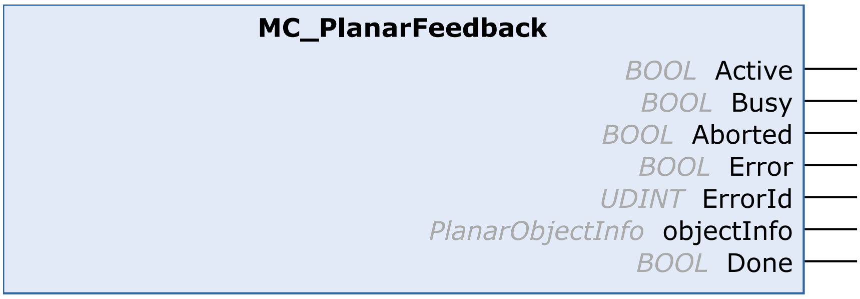
Definition:

```
METHOD GetPlanarObjectInfo : PlanarObjectInfo
```

Return value

PlanarObjectInfo [▶ 128]

7.2.2.2 MC_PlanarFeedback



Displays specific command status information for an associated command, given back by the MC Project.

Syntax

Definition:

```
FUNCTION_BLOCK MC_PlanarFeedback
VAR_OUTPUT
    Active     : BOOL;
    Busy       : BOOL;
    Aborted    : BOOL;
    Error      : BOOL;
    ErrorId    : UDINT;
    objectInfo : PlanarObjectInfo;
    Done       : BOOL;
END_VAR
```

Outputs

Name	Type	Description
Active	BOOL	Indicates an active command, i.e. command was accepted and is being executed.
Busy	BOOL	Indicates a busy command, i.e. command is being processed, waiting for execution, or already executing (= also active).
Aborted	BOOL	Indicates the command is aborted, i.e. execution of the command finished due the start of other commands.
Error	BOOL	Indicates the command has an error.
ErrorId	UDINT	Indicates the error id of the command error.
objectInfo	PlanarObjectInfo [▶ 128]	Indicates which object one would collide with.
Done	BOOL	Indicates the command is done, i.e. execution of the command finished successfully.

Methods

Name	Description
Update [▶ 138]	Updates internal state of the object.

Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.2.1 Update

Updates internal state of the object.

Syntax

Definition:

```
METHOD Update
```

7.2.2.3 MC_PlanarFeedbackBase

Displays general command status information for an associated command, given back by the MC Project.

Do not call the main FB directly. Only use the available methods.

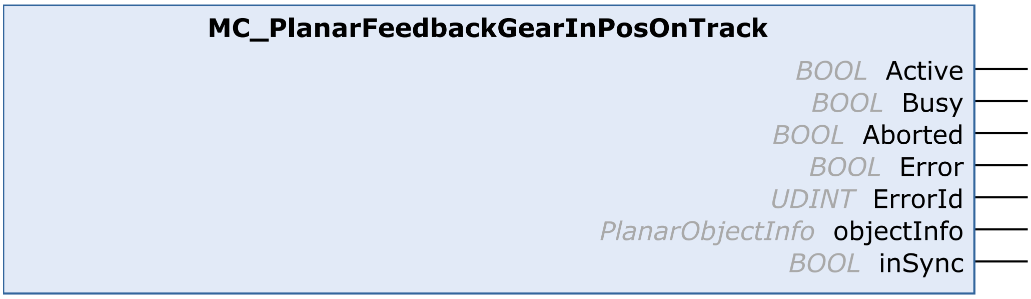
Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.4 MC_PlanarFeedbackGearInPosOnTrack



Displays specific command status information for an associated GearInPosOnTrack command, given back by the MC Project.

Syntax

Definition:

```
FUNCTION_BLOCK MC_PlanarFeedbackGearInPosOnTrack
VAR_OUTPUT
    Active     : BOOL;
    Busy       : BOOL;
    Aborted    : BOOL;
    Error      : BOOL;
    ErrorId    : UDINT;
    objectInfo : PlanarObjectInfo;
    inSync     : BOOL;
END_VAR
```

Outputs

Name	Type	Description
Active	BOOL	Indicates an active command, i.e. command was accepted and is being executed.
Busy	BOOL	Indicates a busy command, i.e. command is being processed, waiting for execution, or already executing (= also active).
Aborted	BOOL	Indicates the command is aborted, i.e. execution of the command finished due the start of other commands.
Error	BOOL	Indicates the command has an error.
ErrorId	UDINT	Indicates the error id of the command error.
objectInfo	PlanarObjectInfo [▶ 128]	Indicates which object one would collide with.
inSync	BOOL	Indicates whether the mover is currently in sync with the master (within tolerance specified in the command options).

Methods

Name	Description
Update [▶ 139]	Updates internal state of the object.

Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.4.1 Update

Update

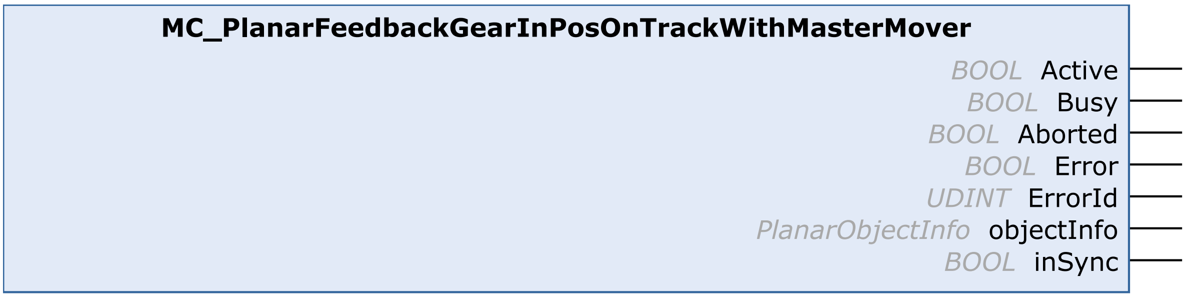
Updates internal state of the object.

Syntax

Definition:

```
METHOD Update
```

7.2.2.5 MC_PlanarFeedbackGearInPosOnTrackWithMasterMover


MC_PlanarFeedbackGearInPosOnTrackWithMasterMover
BOOL Active
BOOL Busy
BOOL Aborted
BOOL Error
UDINT ErrorId
PlanarObjectInfo objectInfo
BOOL inSync

Displays specific command status information for an associated GearInPosOnTrack command, given back by the MC Project.

Syntax

Definition:

```
FUNCTION_BLOCK MC_PlanarFeedbackGearInPosOnTrackWithMasterMover
VAR_OUTPUT
    Active     : BOOL;
    Busy       : BOOL;
    Aborted    : BOOL;
    Error      : BOOL;
    ErrorId    : UDINT;
    objectInfo : PlanarObjectInfo;
    inSync     : BOOL;
END_VAR
```

Outputs

Name	Type	Description
Active	BOOL	Indicates an active command, i.e. command was accepted and is being executed.
Busy	BOOL	Indicates a busy command, i.e. command is being processed, waiting for execution, or already executing (= also active).
Aborted	BOOL	Indicates the command is aborted, i.e. execution of the command finished due the start of other commands.
Error	BOOL	Indicates the command has an error.
ErrorId	UDINT	Indicates the error id of the command error.
objectInfo	PlanarObjectInfo [▸ 128]	Indicates which object one would collide with.
inSync	BOOL	Indicates whether the mover is currently in sync with the master (within tolerance specified in the command options).

Methods

Name	Description
Update [▸ 141]	Updates internal state of the object.

Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.5.1 Update

Updates internal state of the object.

Syntax

Definition:

```
METHOD Update
```

7.2.2.6 MC_PlanarFeedbackInSync

Base class for all specialized feedbacks featuring an inSync output.

Do not call the main FB directly. Only use the available methods.

Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.7 MC_PlanarGroup

A Planar Group object. Planar Movers and other objects added to the group perform collision checks against each other.

Do not call the main FB directly. Only use the available methods.

Methods

Name	Description
Enable [▶ 142]	Starts enabling the Planar Group.
Disable [▶ 142]	Starts disabling the Planar Group.
Reset [▶ 142]	Starts resetting the Planar Group.
Update [▶ 143]	Updates internal state of the object, must be triggered each cycle.

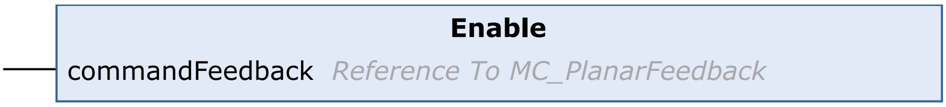
Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.7.1 Enable



Starts enabling the Planar Group.

Syntax

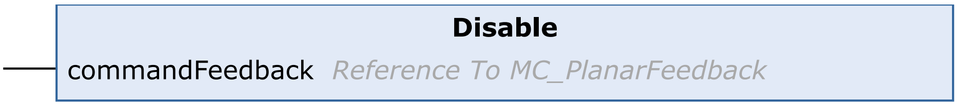
Definition:

```
METHOD Enable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.7.2 Disable



Starts disabling the Planar Group.

Syntax

Definition:

```
METHOD Disable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.7.3 Reset

Reset
commandFeedback
Reference To MC_PlanarFeedback

Starts resetting the Planar Group.

Syntax

Definition:

```
METHOD Reset
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

7.2.2.7.4 Update

Update

Updates internal state of the object, must be triggered each cycle.

Syntax

Definition:

```
METHOD Update
```

7.2.2.8 MC_PlanarMover

A Planar Mover object capable of moving within a plane. Limited movement vertical to the plane is available.

Do not call the main FB directly. Only use the available methods.

Methods

Name	Description
MoveToPosition [▸ 144]	Initiates a direct movement to the specified position.
JoinTrack [▸ 145]	Initiates a direct movement to the specified track. At the end of the movement the mover joins the track.
LeaveTrack [▸ 146]	Initiates a direct movement to the specified position. At the beginning of the movement the track is left.
MoveOnTrack [▸ 146]	Initiates a movement on the track to the specified position and returns command ID.
GearInPosOnTrack [▸ 147]	Initiates a GearInPos movement along a specified trail.
GearInPosOnTrackWithMasterMover [▸ 148]	Initiates a GearInPos movement along a specified trail, in which the master setpoints are provided by another PlanarMover.
MoveZ [▸ 149]	Initiates a movement for the z component.
MoveA [▸ 150]	Initiates a movement for the a component.
MoveB [▸ 150]	Initiates a movement for the b component.
MoveC [▸ 151]	Initiates a movement for the c component.
AdoptTrackOrientation [▸ 151]	Initiates a movement for the c component.
Halt [▸ 152]	Initiates a halt.
Enable [▸ 152]	Starts enabling the Planar Mover.
Disable [▸ 153]	Starts disabling the Planar Mover.
Reset [▸ 153]	Starts resetting the Planar Mover.
Update [▸ 154]	Updates internal state of the object, must be triggered each cycle.

Name	Description
SetPosition [▶ 154]	Sets the position of the Planar Mover. Only possible if the Planar Mover is disabled.
StartExternalSetpointGeneration [▶ 154]	Starts the external setpoint generation, the user must supply setpoints from this PLC cycle on in every PLC cycle.
StopExternalSetpointGeneration [▶ 155]	Ends the external setpoint generation, called after last SetExternalSetpoint (in the same PLC cycle).
SetExternalSetpoint [▶ 155]	Sets the external setpoint for the Planar Mover without ref sys id (for relative mode), must be called each PLC cycle during external setpoint generation.
SetExternalSetpointReferenceId [▶ 156]	Sets the external setpoint for the Planar Mover with corresponing reference system id, must be called each PLC cycle during external setpoint generation.
AddToGroup [▶ 156]	Adds the Planar Mover to the given Planar Group.
RemoveFromGroup [▶ 157]	Removes the Planar Mover from its current Planar Group, i.e. disables collision checks.
GetPositionOnCurrentPart [▶ 157]	Sets the values of the given position to the movers position values on the current part.
GetPlanarObjectInfo [▶ 158]	Returns mover object info (type: mover, id: OID of nc mover).

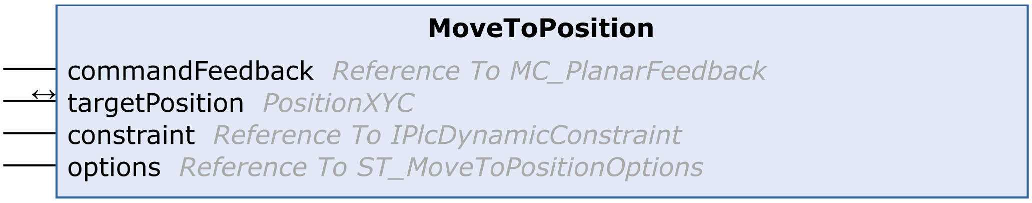
Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.8.1 MoveToPosition


MoveToPosition
commandFeedback Reference To MC_PlanarFeedback
targetPosition PositionXYC
constraint Reference To IPlcDynamicConstraint
options Reference To ST_MoveToPositionOptions

Initiates a direct movement to the specified position.

Syntax

Definition:

```
METHOD MoveToPosition
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    targetPosition  : PositionXYC;
END_VAR
VAR_INPUT
    constraint      : Reference To IPlcDynamicConstraint;
    options         : Reference To ST_MoveToPositionOptions;
END_VAR
```

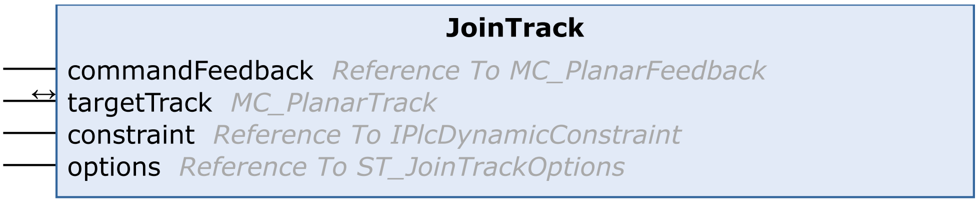
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.
options	Reference To ST_MoveToPositionOptions [▶ 133]	Options for the movement.

In/Outputs

Name	Type	Description
targetPosition	PositionXYC	Target position for the movement.

7.2.2.8.2 JoinTrack


JoinTrack
commandFeedback Reference To MC_PlanarFeedback
targetTrack MC_PlanarTrack
constraint Reference To IPlcDynamicConstraint
options Reference To ST_JoinTrackOptions

Initiates a direct movement to the specified track. At the end of the movement the mover joins the track.

Syntax

Definition:

```
METHOD JoinTrack
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    targetTrack     : MC_PlanarTrack;
END_VAR
VAR_INPUT
    constraint      : Reference To IPlcDynamicConstraint;
    options         : Reference To ST_JoinTrackOptions;
END_VAR
```

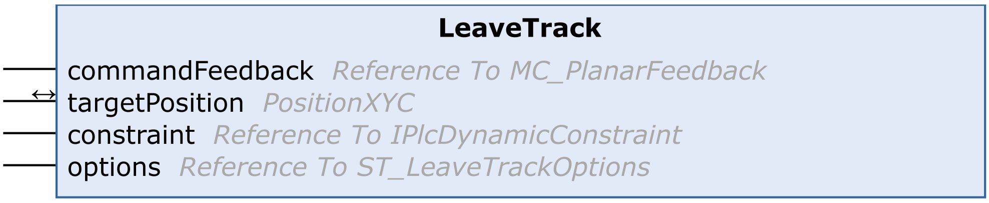
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.
options	Reference To ST_JoinTrackOptions [▶ 131]	Options for the movement.

In/Outputs

Name	Type	Description
targetTrack	MC_PlanarTrack [▶ 161]	Target track for the movement.

7.2.2.8.3 LeaveTrack


LeaveTrack
commandFeedback Reference To MC_PlanarFeedback
targetPosition PositionXYC
constraint Reference To IPlcDynamicConstraint
options Reference To ST_LeaveTrackOptions

Initiates a direct movement to the specified position. At the beginning of the movement the track is left.

Syntax

Definition:

```
METHOD LeaveTrack
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    targetPosition  : PositionXYC;
END_VAR
VAR_INPUT
    constraint      : Reference To IPlcDynamicConstraint;
    options         : Reference To ST_LeaveTrackOptions;
END_VAR
```

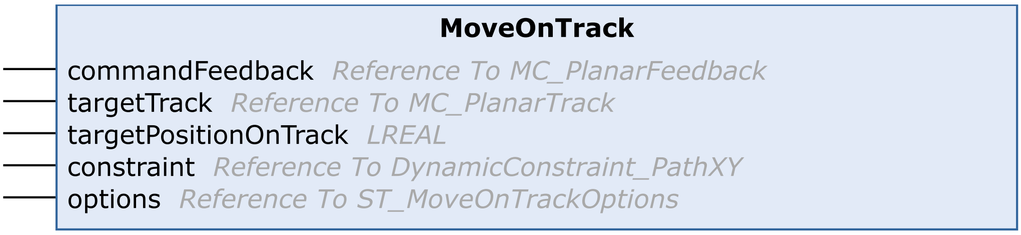
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.
options	Reference To ST_LeaveTrackOptions [▶ 131]	Options for the movement.

In/Outputs

Name	Type	Description
targetPosition	PositionXYC	Target position for the movement.

7.2.2.8.4 MoveOnTrack


MoveOnTrack
commandFeedback Reference To MC_PlanarFeedback
targetTrack Reference To MC_PlanarTrack
targetPositionOnTrack LREAL
constraint Reference To DynamicConstraint_PathXY
options Reference To ST_MoveOnTrackOptions

Initiates a movement on the track to the specified position and returns command ID.

Syntax

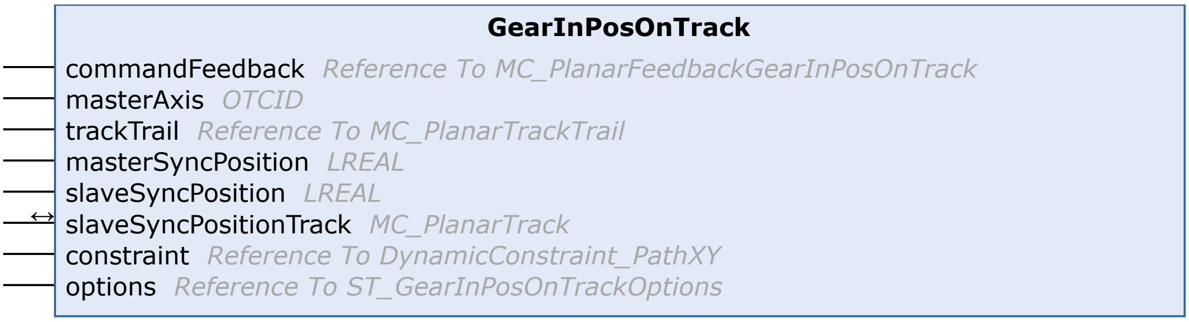
Definition:

```
METHOD MoveOnTrack
VAR_INPUT
    commandFeedback        : Reference To MC_PlanarFeedback;
    targetTrack            : Reference To MC_PlanarTrack;
    targetPositionOnTrack : LREAL;
    constraint             : Reference To DynamicConstraint_PathXY;
    options                : Reference To ST_MoveOnTrackOptions;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
targetTrack	Reference To MC_PlanarTrack [▶ 161]	Target track for the movement. If none is specified, this defaults to the current track.
targetPositionOnTrack	LREAL	Target position on the target track.
constraint	Reference To DynamicConstraint_PathXY	Constraint on maximal dynamics during the movement (V,A,D,J).
options	Reference To ST_MoveOnTrackOptions [▶ 132]	Options for the movement.

7.2.2.8.5 GearInPosOnTrack


GearInPosOnTrack
commandFeedback Reference To MC_PlanarFeedbackGearInPosOnTrack
masterAxis OTCID
trackTrail Reference To MC_PlanarTrackTrail
masterSyncPosition LREAL
slaveSyncPosition LREAL
slaveSyncPositionTrack MC_PlanarTrack
constraint Reference To DynamicConstraint_PathXY
options Reference To ST_GearInPosOnTrackOptions

Initiates a GearInPos movement along a specified trail.

Syntax

Definition:

```
METHOD GearInPosOnTrack
VAR_INPUT
    commandFeedback         : Reference To MC_PlanarFeedbackGearInPosOnTrack;
    masterAxis              : OTCID;
    trackTrail              : Reference To MC_PlanarTrackTrail;
    masterSyncPosition      : LREAL;
    slaveSyncPosition       : LREAL;
END_VAR
VAR_IN_OUT
    slaveSyncPositionTrack : MC_PlanarTrack;
END_VAR
VAR_INPUT
    constraint              : Reference To DynamicConstraint_PathXY;
    options                 : Reference To ST_GearInPosOnTrackOptions;
END_VAR
```

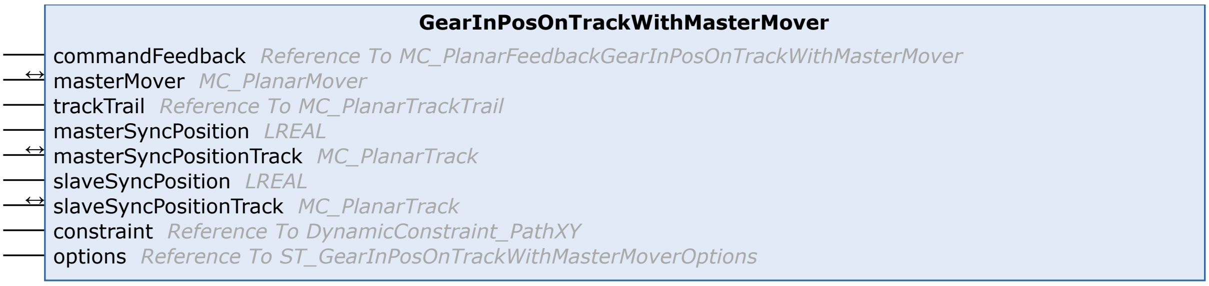
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedbackGearInPosOnTrack [▸ 138]	The command specific feedback object for the command.
masterAxis	OTCID	Master axis being followed.
trackTrail	Reference To MC_PlanarTrackTrail [▸ 171]	Track trail determining along which tracks the GearInPos movement is allowed to proceed.
masterSyncPosition	LREAL	Position of the master axis at which the slave is inSync.
slaveSyncPosition	LREAL	Arc length on track given by slaveSyncPositionTrackOID at which the slave is inSync. Possibly interpreted in modulo fashion, depending on options.
constraint	Reference To DynamicConstraint_PathXY	Constraint on maximal dynamics during the movement (V,A,D,J).
options	Reference To ST_GearInPosOnTrackOptions [▸ 129]	Options for the movement.

In/Outputs

Name	Type	Description
slaveSyncPositionTrack	MC_PlanarTrack [▸ 161]	Track on which the slave is inSync.

7.2.2.8.6 GearInPosOnTrackWithMasterMover


GearInPosOnTrackWithMasterMover
commandFeedback Reference To MC_PlanarFeedbackGearInPosOnTrackWithMasterMover
masterMover MC_PlanarMover
trackTrail Reference To MC_PlanarTrackTrail
masterSyncPosition LREAL
masterSyncPositionTrack MC_PlanarTrack
slaveSyncPosition LREAL
slaveSyncPositionTrack MC_PlanarTrack
constraint Reference To DynamicConstraint_PathXY
options Reference To ST_GearInPosOnTrackWithMasterMoverOptions

Initiates a GearInPos movement along a specified trail, in which the master setpoints are provided by another PlanarMover.

Syntax

Definition:

```
METHOD GearInPosOnTrackWithMasterMover
VAR_INPUT
    commandFeedback         : Reference To MC_PlanarFeedbackGearInPosOnTrackWithMasterMover;
END_VAR
VAR_IN_OUT
    masterMover             : MC_PlanarMover;
END_VAR
VAR_INPUT
    trackTrail              : Reference To MC_PlanarTrackTrail;
    masterSyncPosition      : LREAL;
END_VAR
VAR_IN_OUT
    masterSyncPositionTrack : MC_PlanarTrack;
END_VAR
VAR_INPUT
```

```
    slaveSyncPosition        : LREAL;
END_VAR
VAR_IN_OUT
    slaveSyncPositionTrack   : MC_PlanarTrack;
END_VAR
VAR_INPUT
    constraint               : Reference To DynamicConstraint_PathXY;
    options                  : Reference To ST_GearInPosOnTrackWithMasterMoverOptions;
END_VAR
```

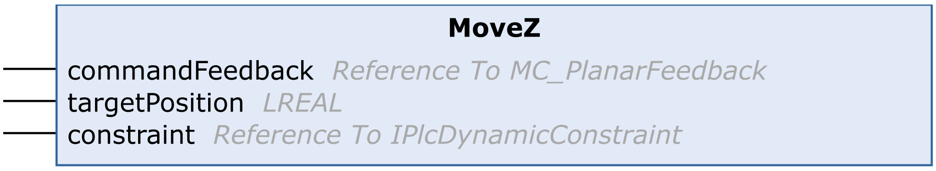
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedbackGearInPosOnTrackWithMasterMover [▸ 140]	The command specific feedback object for the command.
trackTrail	Reference To MC_PlanarTrackTrail [▸ 171]	Track trail determining along which tracks the GearInPos movement is allowed to proceed.
masterSyncPosition	LREAL	Position of the master axis at which the slave is inSync.
slaveSyncPosition	LREAL	Arc length on track given by slaveSyncPositionTrackOID at which the slave is inSync. Possibly interpreted in modulo fashion, depending on options.
constraint	Reference To DynamicConstraint_PathXY	Constraint on maximal dynamics during the movement (V,A,D,J).
options	Reference To ST_GearInPosOnTrackWithMasterMoverOptions [▸ 130]	Options for the movement.

In/Outputs

Name	Type	Description
masterMover	MC_PlanarMover [▸ 143]	Master mover being followed.
masterSyncPositionTrack	MC_PlanarTrack [▸ 161]	Track on which the master is inSync.
slaveSyncPositionTrack	MC_PlanarTrack [▸ 161]	Track on which the slave is inSync.

7.2.2.8.7 MoveZ


MoveZ
commandFeedback Reference To MC_PlanarFeedback
targetPosition LREAL
constraint Reference To IPlcDynamicConstraint

Initiates a movement for the z component.

Syntax

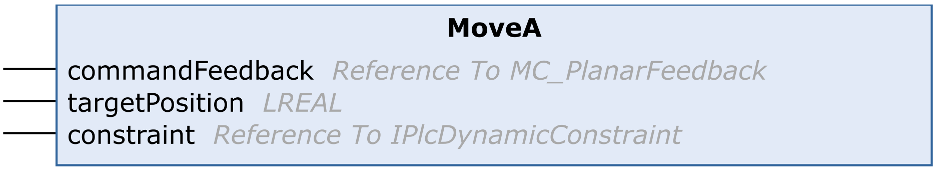
Definition:

```
METHOD MoveZ
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    targetPosition  : LREAL;
    constraint      : Reference To IPlcDynamicConstraint;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
targetPosition	LREAL	Target position for the movement.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.

7.2.2.8.8 MoveA


MoveA
commandFeedback Reference To MC_PlanarFeedback
targetPosition LREAL
constraint Reference To IPlcDynamicConstraint

Initiates a movement for the a component.

Syntax

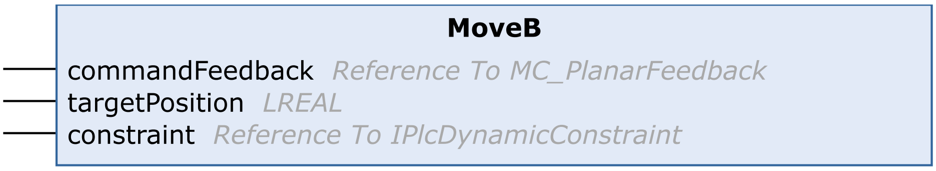
Definition:

```
METHOD MoveA
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    targetPosition  : LREAL;
    constraint      : Reference To IPlcDynamicConstraint;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
targetPosition	LREAL	Target position for the movement.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.

7.2.2.8.9 MoveB


MoveB
commandFeedback Reference To MC_PlanarFeedback
targetPosition LREAL
constraint Reference To IPlcDynamicConstraint

Initiates a movement for the b component.

Syntax

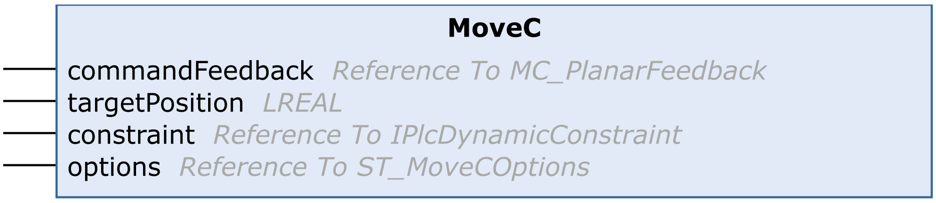
Definition:

```
METHOD MoveB
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    targetPosition  : LREAL;
    constraint      : Reference To IPlcDynamicConstraint;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
targetPosition	LREAL	Target position for the movement.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.

7.2.2.8.10 MoveC


MoveC
commandFeedback Reference To MC_PlanarFeedback
targetPosition LREAL
constraint Reference To IPlcDynamicConstraint
options Reference To ST_MoveCOptions

Initiates a movement for the c component.

Syntax

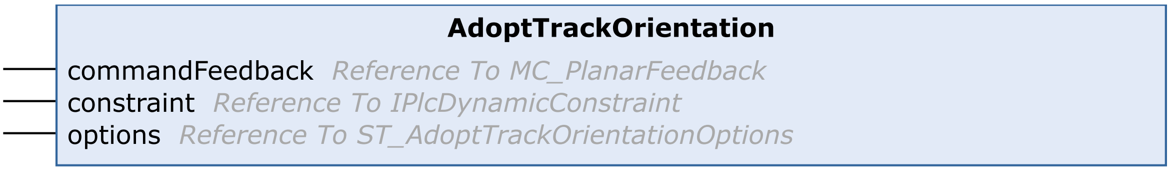
Definition:

```
METHOD MoveC
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    targetPosition  : LREAL;
    constraint      : Reference To IPlcDynamicConstraint;
    options         : Reference To ST_MoveCOptions;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
targetPosition	LREAL	Target position for the movement.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.
options	Reference To ST_MoveCOptions [▶ 132]	Options for the rotation.

7.2.2.8.11 AdoptTrackOrientation


AdoptTrackOrientation
commandFeedback Reference To MC_PlanarFeedback
constraint Reference To IPlcDynamicConstraint
options Reference To ST_AdoptTrackOrientationOptions

Initiates a movement for the c component.

Syntax

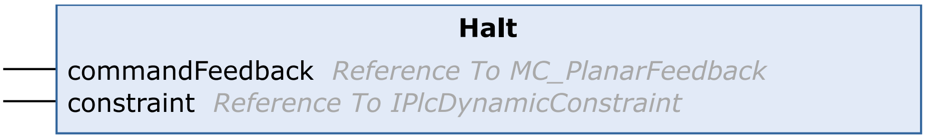
Definition:

```
METHOD AdoptTrackOrientation
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    constraint      : Reference To IPlcDynamicConstraint;
    options         : Reference To ST_AdoptTrackOrientationOptions;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.
options	Reference To ST_AdoptTrackOrientationOptions [▶ 128]	Options for the rotation.

7.2.2.8.12 Halt


Halt
commandFeedback Reference To MC_PlanarFeedback
constraint Reference To IPlcDynamicConstraint

Initiates a halt.

Syntax

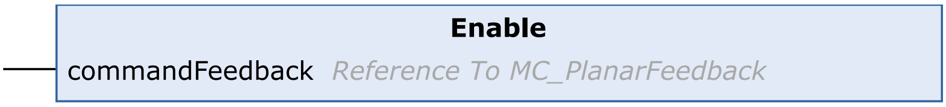
Definition:

```
METHOD Halt
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    constraint      : Reference To IPlcDynamicConstraint;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
constraint	Reference To IPlcDynamicConstraint	Dynamic constraints for this movement.

7.2.2.8.13 Enable



Starts enabling the Planar Mover.

Syntax

Definition:

```
METHOD Enable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.8.14 Disable

Disable
commandFeedback
Reference To MC_PlanarFeedback

Starts disabling the Planar Mover.

Syntax

Definition:

```
METHOD Disable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.8.15 Reset

Reset
commandFeedback
Reference To MC_PlanarFeedback

Starts resetting the Planar Mover.

Syntax

Definition:

```
METHOD Reset
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.8.16 Update

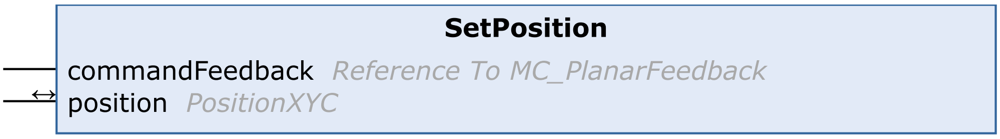
Updates internal state of the object, must be triggered each cycle.

Syntax

Definition:

```
METHOD Update
```

7.2.2.8.17 SetPosition



Sets the position of the Planar Mover. Only possible if the Planar Mover is disabled.

Syntax

Definition:

```
METHOD SetPosition
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    position        : PositionXYC;
END_VAR
```

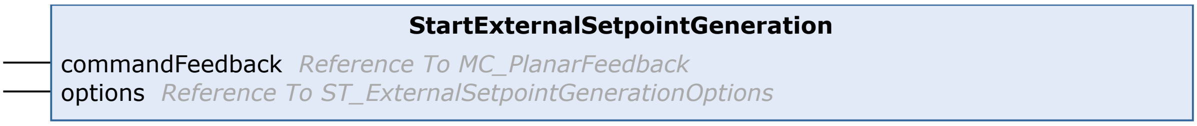
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [⏵ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
position	PositionXYC	New position of the Planar Mover.

7.2.2.8.18 StartExternalSetpointGeneration


StartExternalSetpointGeneration
commandFeedback Reference To MC_PlanarFeedback
options Reference To ST_ExternalSetpointGenerationOptions

Starts the external setpoint generation, the user must supply setpoints from this PLC cycle on in every PLC cycle.

Syntax

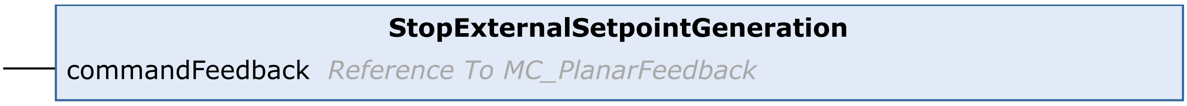
Definition:

```
METHOD StartExternalSetpointGeneration
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    options         : Reference To ST_ExternalSetpointGenerationOptions;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
options	Reference To ST_ExternalSetpointGenerationOptions [▶ 129]	Options for the movement.

7.2.2.8.19 StopExternalSetpointGeneration


StopExternalSetpointGeneration
commandFeedback Reference To MC_PlanarFeedback

Ends the external setpoint generation, called after last SetExternalSetpoint (in the same PLC cycle).

Syntax

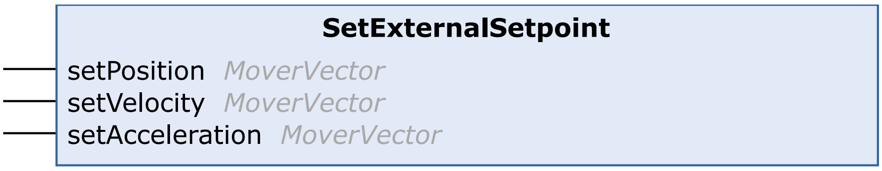
Definition:

```
METHOD StopExternalSetpointGeneration
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.8.20 SetExternalSetpoint



Sets the external setpoint for the Planar Mover without ref sys id (for relative mode), must be called each PLC cycle during external setpoint generation.

Syntax

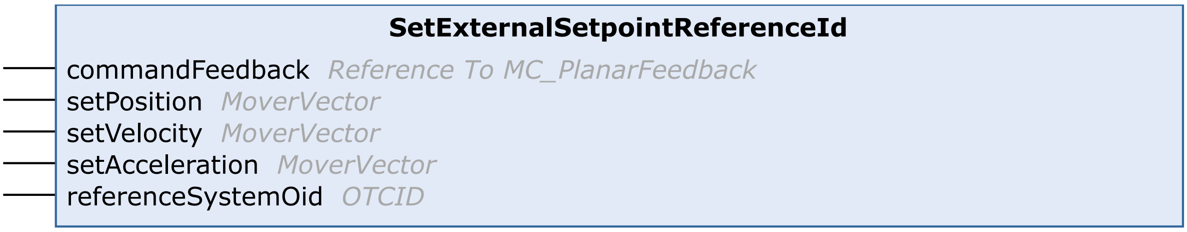
Definition:

```
METHOD SetExternalSetpoint
VAR_INPUT
    setPosition     : MoverVector;
    setVelocity     : MoverVector;
    setAcceleration : MoverVector;
END_VAR
```

Inputs

Name	Type	Description
setPosition	MoverVector	Position that is send to the Planar Mover.
setVelocity	MoverVector	Velocity that is send to the Planar Mover.
setAcceleration	MoverVector	Acceleration that is send to the Planar Mover.

7.2.2.8.21 SetExternalSetpointReferenceId


SetExternalSetpointReferenceId
commandFeedback Reference To MC_PlanarFeedback
setPosition MoverVector
setVelocity MoverVector
setAcceleration MoverVector
referenceSystemOid OTCID

Sets the external setpoint for the Planar Mover with corresponing reference system id, must be called each PLC cycle during external setpoint generation.

Syntax

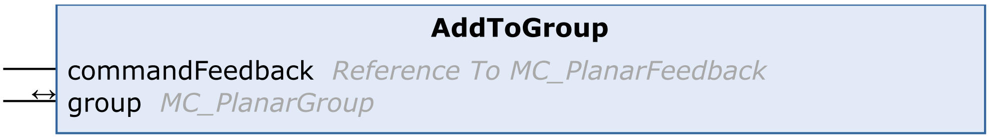
Definition:

```
METHOD SetExternalSetpointReferenceId
VAR_INPUT
    commandFeedback    : Reference To MC_PlanarFeedback;
    setPosition        : MoverVector;
    setVelocity        : MoverVector;
    setAcceleration    : MoverVector;
    referenceSystemOid : OTCID;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
setPosition	MoverVector	Position that is send to the Planar Mover.
setVelocity	MoverVector	Velocity that is send to the Planar Mover.
setAcceleration	MoverVector	Acceleration that is send to the Planar Mover.
referenceSystemOid	OTCID	Part or coordinate system id.

7.2.2.8.22 AddToGroup



Adds the Planar Mover to the given Planar Group.

Syntax

Definition:

```
METHOD AddToGroup
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

```
VAR_IN_OUT
    group             : MC_PlanarGroup;
END_VAR
```

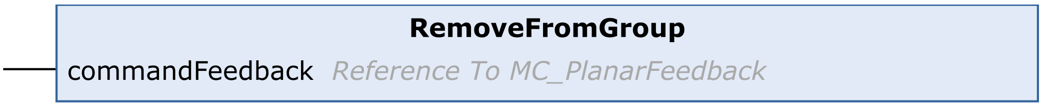
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
group	MC_PlanarGroup [▶ 141]	The Planar Group that the Planar Mover joins.

7.2.2.8.23 RemoveFromGroup



Removes the Planar Mover from its current Planar Group, i.e. disables collision checks.

Syntax

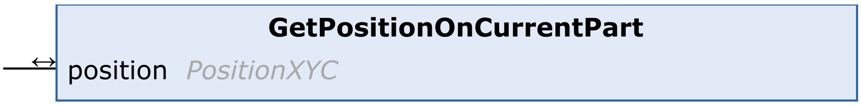
Definition:

```
METHOD RemoveFromGroup
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.8.24 GetPositionOnCurrentPart



Sets the values of the given position to the movers position values on the current part.

Syntax

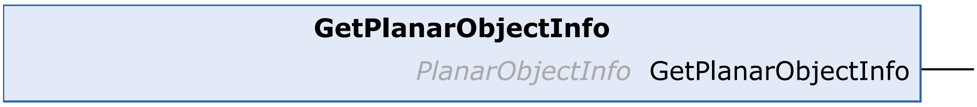
Definition:

```
METHOD GetPositionOnCurrentPart
VAR_IN_OUT
    position : PositionXYC;
END_VAR
```

In/Outputs

Name	Type	Description
position	PositionXYC	The position on the planar part.

7.2.2.8.25 GetPlanarObjectInfo



Returns mover object info (type: mover, id: OID of nc mover).

Syntax

Definition:

```
METHOD GetPlanarObjectInfo : PlanarObjectInfo
```

Return value

PlanarObjectInfo [▶ 128]

7.2.2.9 MC_PlanarPart

A Planar Part object represents the area that Planar Movers can move on. It contains information about the stator objects.

Do not call the main FB directly. Only use the available methods.

Methods

Name	Description
Initialize [▶ 159]	Initialize the Planar Part, i.e. connecting it to the MC via its OID.
ActivatePosition [▶ 159]	Activates the position given by part position index.
AllowEnable [▶ 160]	From now on the part can be enabled until ForceDisablePart is called.
ForceDisable [▶ 160]	Disables the part and keeps it disabled until AllowEnabledPart is called.
Reset [▶ 160]	Resets the part.
GetPosition [▶ 161]	Sets the values of the given position to the parts position values.

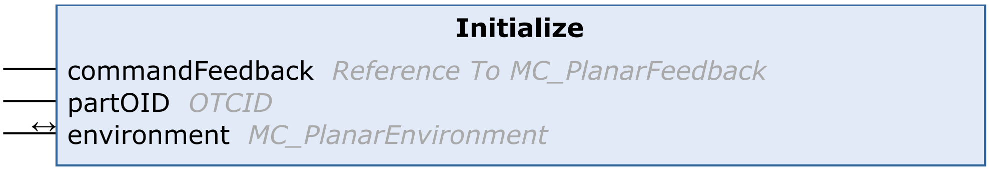
Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.40 Advanced Motion Pack V3.2.60	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.9.1 Initialize


Initialize
commandFeedback Reference To MC_PlanarFeedback
partOID OTCID
environment MC_PlanarEnvironment

Initialize the Planar Part, i.e. connecting it to the MC via its OID.

Syntax

Definition:

```
METHOD Initialize
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    partOID         : OTCID;
END_VAR
VAR_IN_OUT
    environment     : MC_PlanarEnvironment;
END_VAR
```

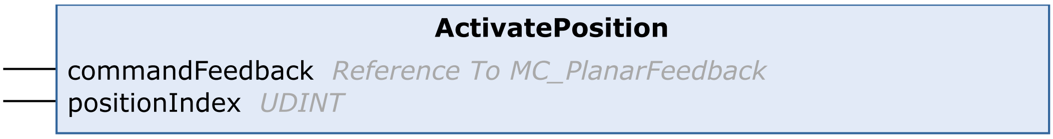
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.
partOID	OTCID	OID of the part.

In/Outputs

Name	Type	Description
environment	MC_PlanarEnvironment [▸ 134]	Environment the part is in.

7.2.2.9.2 ActivatePosition



Activates the position given by part position index.

Syntax

Definition:

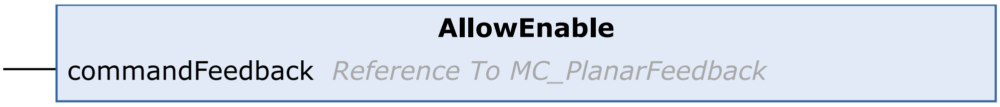
```
METHOD ActivatePosition
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
    positionIndex   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

Name	Type	Description
positionIndex	UDINT	Index of the position.

7.2.2.9.3 AllowEnable



From now on the part can be enabled until ForceDisablePart is called.

Syntax

Definition:

```
METHOD AllowEnable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.9.4 ForceDisable

ForceDisable
commandFeedback Reference To MC_PlanarFeedback

Disables the part and keeps it disabled until AllowEnabledPart is called.

Syntax

Definition:

```
METHOD ForceDisable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.9.5 Reset

Reset
commandFeedback Reference To MC_PlanarFeedback

Resets the part.

Syntax

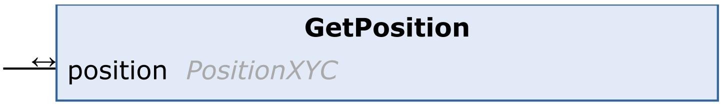
Definition:

```
METHOD Reset
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

7.2.2.9.6 GetPosition



Sets the values of the given position to the parts position values.

Syntax

Definition:

```
METHOD GetPosition
VAR_IN_OUT
    position : PositionXYC;
END_VAR
```

In/Outputs

Name	Type	Description
position	PositionXYC	The position of the Planar Part.

7.2.2.10 MC_PlanarTrack

A track within a plane which Planar Movers can follow. Planar Movers on the track automatically avoid collisions with each other. The Planar Track can consist of several consecutive segments and be joined with other Planar Tracks at its start/end.

Do not call the main FB directly. Only use the available methods.

Methods

Name	Description
Clear [▸ 162]	Clears the geometric information of the Planar Track.
AppendPosition [▸ 163]	Appends a position to the Planar Track.
AppendLine [▸ 163]	Appends a line to the Planar Track.
AppendCircle [▸ 164]	Appends a circular arc to the Planar Track.
CloseLoop [▸ 164]	Closes the loop of the Planar Track, no further part can be appended.
StartFromTrack [▸ 165]	Sets the other Planar Track's endpoint as start point of this Planar Track, transition is smooth. The other Planar Track is blocked for further changes (until it is cleared).

Name	Description
EndAtTrack [▶ 165]	Appends a smooth transition from the end of this Planar Track to the other Planar Track's start point. The Planar Track is blocked for further changes (until it is cleared).
StartFromTrackAdvanced [▶ 166]	Sets the other Planar Track's endpoint as start point of this Planar Track, transition is smooth. The other Planar Track is blocked for further geometrical changes (until it is cleared).
EndAtTrackAdvanced [▶ 167]	Appends a smooth transition from the end of this Planar Track to the other Planar Track's start point. The Planar Track is blocked for further geometrical changes (until it is cleared).
Enable [▶ 167]	Starts enabling the Planar Track.
Disable [▶ 168]	Starts disabling the Planar Track.
Reset [▶ 168]	Starts resetting the Planar Track.
GetArcLengthClosestTo [▶ 168]	Calculate the arc length value where the Planar Track is closest to a geometry's center point.
GetPositionAt [▶ 169]	Get a position on the Planar Track at a specific arc length value.
GetLength [▶ 169]	Returns the Planar Track's length, -1 return value indicates no connection to Nc Track.
GetPlanarObjectInfo [▶ 170]	Returns track object info (type: track, id: OID of nc track).
Update [▶ 170]	Updates internal state of the object, must be triggered each cycle.
AddToGroup [▶ 170]	Adds the Planar Track to the given Planar Group.
RemoveFromGroup [▶ 171]	Removes the Planar Track from its current Planar Group, i.e. disables collision checks.

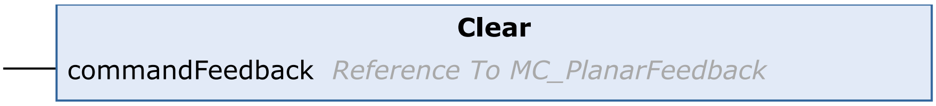
Required License

TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12 Advanced Motion Pack V3.1.10.11	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

7.2.2.10.1 Clear



Clears the geometric information of the Planar Track.

Syntax

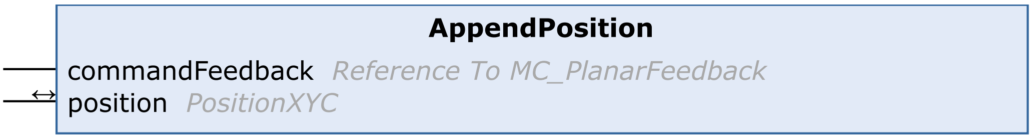
Definition:

```
METHOD Clear
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.10.2 AppendPosition



Appends a position to the Planar Track.

Syntax

Definition:

```
METHOD AppendPosition
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    position        : PositionXYC;
END_VAR
```

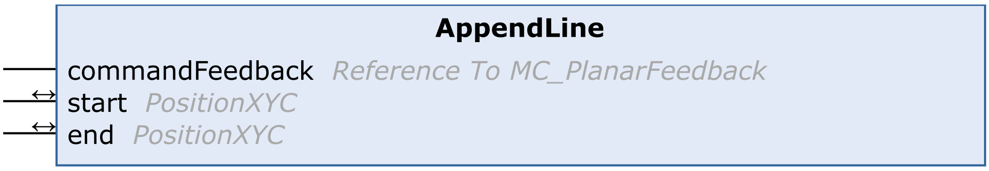
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
position	PositionXYC	Position that is the new endpoint of the Planar Track.

7.2.2.10.3 AppendLine



Appends a line to the Planar Track.

Syntax

Definition:

```
METHOD AppendLine
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    start           : PositionXYC;
    end             : PositionXYC;
END_VAR
```

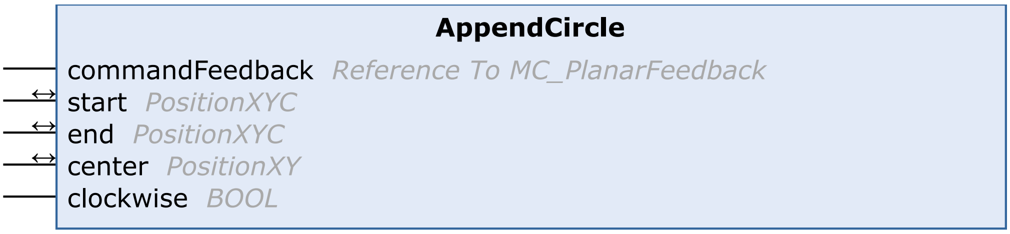
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
start	PositionXYC	Start position of the line.
end	PositionXYC	End position of the line, this position is the new endpoint of the Planar Track.

7.2.2.10.4 AppendCircle



Appends a circular arc to the Planar Track.

Syntax

Definition:

```
METHOD AppendCircle
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    start           : PositionXYC;
    end             : PositionXYC;
    center          : PositionXY;
END_VAR
VAR_INPUT
    clockwise       : BOOL;
END_VAR
```

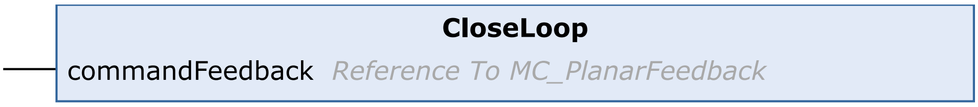
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
clockwise	BOOL	Indicates if the clockwise circle is appended.

In/Outputs

Name	Type	Description
start	PositionXYC	Start position of the circular arc.
end	PositionXYC	End position of the circular arc, this position is the new endpoint of the Planar Track.
center	PositionXY	Center of the circular arc.

7.2.2.10.5 CloseLoop



Closes the loop of the Planar Track, no further part can be appended.

Syntax

Definition:

```
METHOD CloseLoop
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

7.2.2.10.6 StartFromTrack

StartFromTrack
commandFeedback Reference To MC_PlanarFeedback
↔ track MC_PlanarTrack

Sets the other Planar Track's endpoint as start point of this Planar Track, transition is smooth. The other Planar Track is blocked for further changes (until it is cleared).

Syntax

Definition:

```
METHOD StartFromTrack
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    track           : MC_PlanarTrack;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▸ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
track	MC_PlanarTrack [▸ 161]	The other Planar Track.

7.2.2.10.7 EndAtTrack

EndAtTrack
commandFeedback Reference To MC_PlanarFeedback
↔ track MC_PlanarTrack

Appends a smooth transition from the end of this Planar Track to the other Planar Track's start point. The Planar Track is blocked for further changes (until it is cleared).

Syntax

Definition:

```
METHOD EndAtTrack
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    track           : MC_PlanarTrack;
END_VAR
```

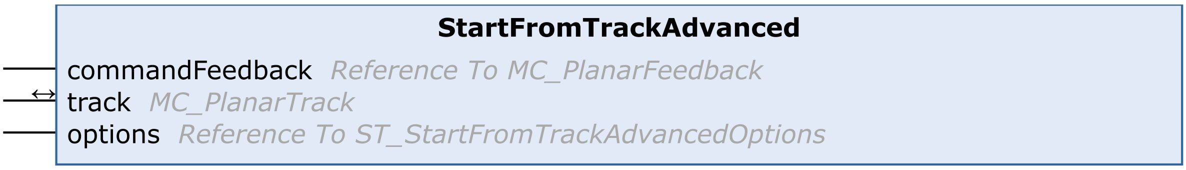
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
track	MC_PlanarTrack [▶ 161]	The other Planar Track.

7.2.2.10.8 StartFromTrackAdvanced


StartFromTrackAdvanced
commandFeedback Reference To MC_PlanarFeedback
track MC_PlanarTrack
options Reference To ST_StartFromTrackAdvancedOptions

Sets the other Planar Track's endpoint as start point of this Planar Track, transition is smooth. The other Planar Track is blocked for further geometrical changes (until it is cleared).

Syntax

Definition:

```
METHOD StartFromTrackAdvanced
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    track           : MC_PlanarTrack;
END_VAR
VAR_INPUT
    options         : Reference To ST_StartFromTrackAdvancedOptions;
END_VAR
```

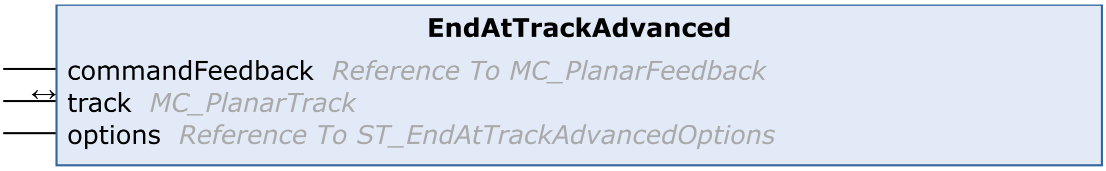
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
options	Reference To ST_StartFromTrackAdvancedOptions [▶ 133]	Options for the connection, i.e. which connections are closed.

In/Outputs

Name	Type	Description
track	MC_PlanarTrack [▶ 161]	The other Planar Track.

7.2.2.10.9 EndAtTrackAdvanced


EndAtTrackAdvanced

commandFeedback *Reference To MC_PlanarFeedback*
↔ track *MC_PlanarTrack*
options *Reference To ST_EndAtTrackAdvancedOptions*

Appends a smooth transition from the end of this Planar Track to the other Planar Track's start point. The Planar Track is blocked for further geometrical changes (until it is cleared).

Syntax

Definition:

```
METHOD EndAtTrackAdvanced
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    track           : MC_PlanarTrack;
END_VAR
VAR_INPUT
    options         : Reference To ST_EndAtTrackAdvancedOptions;
END_VAR
```

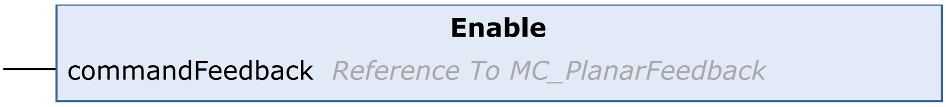
Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.
options	Reference To ST_EndAtTrackAdvancedOptions [▶ 129]	Options for the connection, i.e. which connections are closed.

In/Outputs

Name	Type	Description
track	MC_PlanarTrack [▶ 161]	The other Planar Track.

7.2.2.10.10 Enable


Enable

commandFeedback *Reference To MC_PlanarFeedback*

Starts enabling the Planar Track.

Syntax

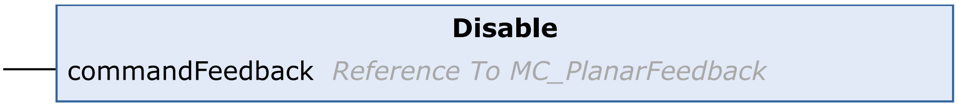
Definition:

```
METHOD Enable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.10.11 Disable



Starts disabling the Planar Track.

Syntax

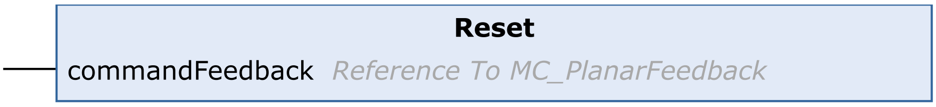
Definition:

```
METHOD Disable
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.10.12 Reset



Starts resetting the Planar Track.

Syntax

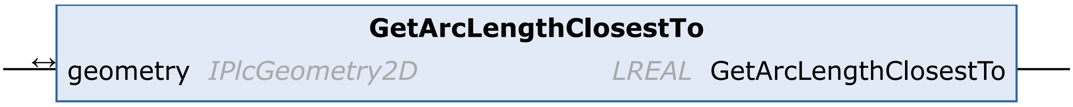
Definition:

```
METHOD Reset
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.10.13 GetArcLengthClosestTo



Calculate the arc length value where the Planar Track is closest to a geometry's center point.

Syntax

Definition:

```
METHOD GetArcLengthClosestTo : LREAL
VAR_IN_OUT
    geometry : IPlcGeometry2D;
END_VAR
```

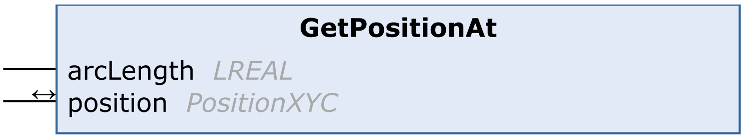
In/Outputs

Name	Type	Description
geometry	IPlcGeometry2D	The geometry to check the arc length for.

Return value

LREAL

7.2.2.10.14 GetPositionAt



Get a position on the Planar Track at a specific arc length value.

Syntax

Definition:

```
METHOD GetPositionAt
VAR_INPUT
    arcLength : LREAL;
END_VAR
VAR_IN_OUT
    position  : PositionXYC;
END_VAR
```

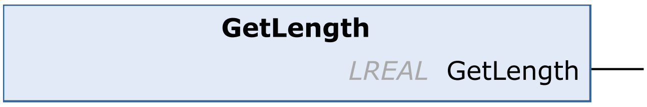
Inputs

Name	Type	Description
arcLength	LREAL	Arc length value where the position is evaluated.

In/Outputs

Name	Type	Description
position	PositionXYC	The position at the specified arc parameter.

7.2.2.10.15 GetLength



Returns the Planar Track's length, -1 return value indicates no connection to Nc Track.

Syntax

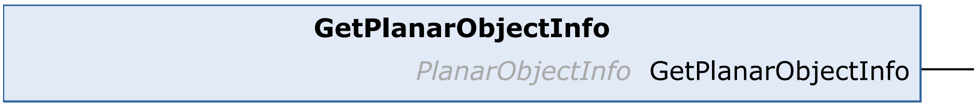
Definition:

```
METHOD GetLength : LREAL
```

Return value

LREAL

7.2.2.10.16 GetPlanarObjectInfo



Returns track object info (type: track, id: OID of nc track).

Syntax

Definition:

```
METHOD GetPlanarObjectInfo : PlanarObjectInfo
```

Return value

PlanarObjectInfo [▶ 128]

7.2.2.10.17 Update

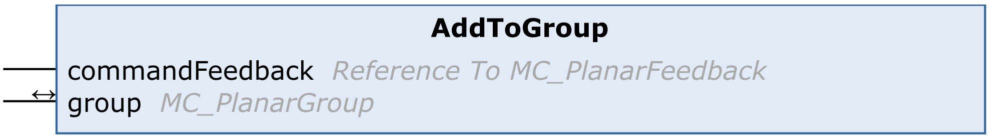
Updates internal state of the object, must be triggered each cycle.

Syntax

Definition:

```
METHOD Update
```

7.2.2.10.18 AddToGroup



Adds the Planar Track to the given Planar Group.

Syntax

Definition:

```
METHOD AddToGroup
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    group           : MC_PlanarGroup;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

In/Outputs

Name	Type	Description
group	MC_PlanarGroup [▶ 141]	The Planar Group that the mover joins.

7.2.2.10.19 RemoveFromGroup

RemoveFromGroup
commandFeedback
Reference To MC_PlanarFeedback

Removes the Planar Track from its current Planar Group, i.e. disables collision checks.

Syntax

Definition:

```
METHOD RemoveFromGroup
VAR_INPUT
    commandFeedback : Reference To MC_PlanarFeedback;
END_VAR
```

Inputs

Name	Type	Description
commandFeedback	Reference To MC_PlanarFeedback [▶ 137]	The feedback object for the command.

7.2.2.11 MC_PlanarTrackTrail

A list of distinct tracks each starting at the ending vertex of its predecessor.

Do not call the main FB directly. Only use the available methods.

Methods

Name	Description
Clear [▶ 172]	Clears the TrackTrail.
AddTrack [▶ 172]	Adds a track to the TrackTrail. The track should start at the end vertex of the currently last track.

Required License

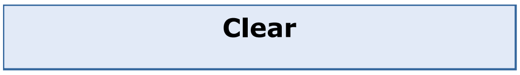
TC3 Planar Motion Base

System Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT V3.1.4024.12	PC or CX (x64)	Tc3_Mc3PlanarMotion, Tc3_Physics

Development environment	Target system type	PLC libraries to include
Advanced Motion Pack V3.1.10.11		

7.2.2.11.1 Clear



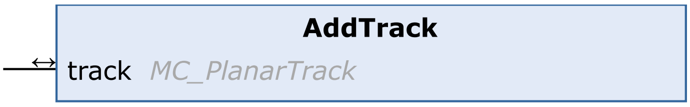
Clears the TrackTrail.

Syntax

Definition:

```
METHOD Clear
```

7.2.2.11.2 AddTrack



Adds a track to the TrackTrail. The track should start at the end vertex of the currently last track.

Syntax

Definition:

```
METHOD AddTrack
VAR_IN_OUT
    track : MC_PlanarTrack;
END_VAR
```

In/Outputs

Name	Type	Description
track	MC_PlanarTrack [▶ 161]	The track to be added to the end of the TrackTrail.

8 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963-157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963-460
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963-0
e-mail:	info@beckhoff.com
web:	www.beckhoff.com

More Information:
www.beckhoff.com/TF5430

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