# Table of contents

1 Foreword ....................................................................................................................... 5  
  1.1 Notes on the documentation ....................................................................................... 5  
  1.2 Safety instructions ...................................................................................................... 6  
  1.3 Notes on information security .................................................................................... 7  

2 Introduction ................................................................................................................... 8  

3 TF5430 TwinCAT 3 Planar Motion – overview of the new features .................................. 9  

4 States and modes .......................................................................................................... 10  
  4.1 Planar objects state diagram ..................................................................................... 10  
  4.2 Planar mover command diagram ............................................................................... 11  
  4.3 Planar track operation modes .................................................................................. 12  

5 Planar Motion components ........................................................................................... 14  
  5.1 Planar mover ............................................................................................................ 14  
    5.1.1 Configuration ....................................................................................................... 14  
    5.1.2 Creating a PLC .................................................................................................... 17  
    5.1.3 Example: "Creating and moving Planar movers" ............................................... 18  
    5.1.4 Example: "Creating and moving a Planar mover with auxiliary axes" ............. 21  
    5.1.5 Example "Creating and moving a Planar mover with External Setpoint Generation" .... 23  
    5.1.6 Example "Moving the Planar mover in CRotationFreeMovement mode" .......... 26  
    5.1.7 Limits and options of the motion commands ..................................................... 27  
  5.2 Planar track .............................................................................................................. 29  
    5.2.1 Configuration ....................................................................................................... 29  
    5.2.2 Track networks and collision avoidance ............................................................ 32  
    5.2.3 Example "Joining and moving a Planar mover on the track" .............................. 35  
    5.2.4 Example "Coupling a Planar mover to a track and moving it in CRotationOnTrack mode" ................................................................. 38  
    5.2.5 Example "Coupling a Planar mover to a track and moving it with AdoptTrackOrientation" ................................................................. 41  
    5.2.6 Example "Synchronizing a Planar mover on a track with one axis" ................. 44  
    5.2.7 Example: "Synchronizing a Planar mover on a track with another Planar mover" .... 51  
    5.2.8 Example "Connecting Planar tracks to a network" ............................................ 54  
    5.2.9 Example: "Following a Planar mover through a Track Network" ..................... 58  
  5.3 Planar group ............................................................................................................ 62  
    5.3.1 Configuration ....................................................................................................... 62  
    5.3.2 Example: "Creating and moving Planar movers with group" ............................ 65  
  5.4 Planar environment ................................................................................................. 69  
    5.4.1 Configuration ....................................................................................................... 69  
    5.4.2 Example "Configuring the stator area and boundary" ...................................... 71  
  5.5 Example: "Creating and moving Planar movers with track and group" ..................... 74  
  5.6 Planar Feedback ...................................................................................................... 77  
    5.6.1 Example "Creating a Planar mover and Planar Feedback" ............................... 78  
    5.6.2 Example "Planar motion components: averting collision" ............................... 80  
    5.6.3 Specialized feedback types .................................................................................. 84  
  5.7 Planar TrackTrail ...................................................................................................... 85  
    5.7.1 Example "Synchronization movement over two Planar tracks" ....................... 86
# Table of contents

6 PLC Libraries ................................................................................................................................. 90
  6.1 Inserting libraries ...................................................................................................................... 90
  6.2 Tc3_Mc3PlanarMotion API ....................................................................................................... 91
    6.2.1 Data Types ....................................................................................................................... 91
    6.2.2 Function Blocks ................................................................................................................. 98
1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards. It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components. It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development. We reserve the right to revise and change the documentation at any time and without prior announcement. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH. Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany. The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>⚠️ DANGER</td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td>⚠️ WARNING</td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td>⚠️ CAUTION</td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td>📌 NOTE</td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

- **Tip or pointer**

This symbol indicates information that contributes to better understanding.
1.3  Notes on information security

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

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our [https://www.beckhoff.com/secguide](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](https://www.beckhoff.com/secinfo).
2 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.
3 TF5430 TwinCAT 3 Planar Motion – overview of the new features

From version V3.1.10.51:
- New: AdoptTrackOrientation rotates the mover on the track to 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
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.

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.
Planar Motion components

5.1 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.

5.1.1 Configuration

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

1. Select MOTION > Add New Item….

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

3. Select MC Project > Axes > Add New Item….
4. In the following dialog box, create one (or more) Planar movers and confirm with **OK**.

The Planar mover is now created and can be parameterized.

**Open detailed description**
- Select the Planar mover in the tree and double-click it.

**Meaning of the individual tabs**

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

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.

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. The "Initial Position" parameter should be set in simulation mode. If the real mover has no default dimensions, the "Mover width" and "Mover height" parameters must be adjusted.

From version V3.1.10.30: The "Minimum/Maximum Position" parameters are used to define for the C axis when the mover switches to the CRotation command mode. For all target positions of C-movements, the parameters "C coordinate modulus" and "C coordinate modulo tolerance window" (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 [41].
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") as well as state information are displayed.

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

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

**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.

5.1.2 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**.

3. Add the libraries "Tc3_Physics" and "Tc3_Mc3PlanarMotion" to the PLC project; see Inserting libraries [p. 90].
   ⇨ The PLC is created and you can issue commands to the corresponding objects as described in the following examples.

### 5.1.3  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 [p. 14].

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 under **Creating a PLC** [p. 17].

3. Use **MAIN** to create the mover or movers ("MC_PlanarMover") as follows.
They represent the movers in the MC Configuration.

4. 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.

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

5. Then program a sequence in MAIN.

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

```plaintext
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

6. 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.

7. 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

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

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

10. Log in the PLC via the button in the menu bar.
11. 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.
5.1.4 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 [14].

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 [17].

3. Use MAIN to create the mover or movers ("MC_PlanarMover") as follows.

```plaintext
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

5. 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" [26]
- Limits and options of the motion commands [27]

Sending the command

6. 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.

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

   ![Link To PLC... button](image)

   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

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

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

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

11. 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.
5.1.5 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 [14].
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 [17].
3. Use MAIN to create the mover or movers ("MC_PlanarMover") as follows.

They represent the movers in the MC Configuration.
4. Create a Planar mover, a state variable for a state machine and variables for the external setpoint, as shown below.

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

5. 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).

```plaintext
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:
    p.x := p.x + deltat * v.x;
    v.x := v.x + deltat * a.x;
    a.x := a.x + deltat * 1.0;
    mover.SetExternalSetpoint(p,v,a);
    IF a.x >= 1.0 THEN
      state := 4;
    END_IF;
  4:
    p.x := p.x + deltat * v.x;
    v.x := v.x + deltat * a.x;
    a.x := a.x + deltat * 1.0;
    mover.SetExternalSetpoint(p,v,a);
    IF a.x >= 1.0 THEN
      state := 5;
    END_IF;
  5:
    mover.StopExternalSetpointGeneration(0);
    state := 6;
END_CASE
```

**Sending the command**

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

```plaintext
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.

7. 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.
Planar Motion components

TF5430

Version: 1.3

Activating and starting the project

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

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

10. Log in the PLC via the button in the menu bar.
11. 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.
5.1.6 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 [14].
- 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 [17].
- 3. Use MAIN to create the mover or movers ("MC_PlanarMover") as follows.

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

5. 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

- 6. 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.
7. 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**

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

9. Set the TwinCAT system to the "Run" state via the **button.**

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

11. 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.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
<th>Prepared value</th>
<th>Address</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>mover</td>
<td>NC_PlanarMover</td>
<td>LREAL</td>
<td>0</td>
<td>REFERENCE TO...</td>
<td>Move mover data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td></td>
<td>CDT_PLCOTMPC</td>
<td></td>
<td></td>
<td>%IB*</td>
<td>Move data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td></td>
<td>CDT_MCTOPLC</td>
<td></td>
<td></td>
<td>%IB*</td>
<td>Move standard data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td>MoverID</td>
<td>OTCID</td>
<td>16#05110010</td>
<td></td>
<td></td>
<td>Object id of the planar mover.</td>
</tr>
<tr>
<td>GroupID</td>
<td>OTCID</td>
<td>16#00000000</td>
<td></td>
<td></td>
<td>Object id of the planar group the mover is in.</td>
</tr>
<tr>
<td>State</td>
<td>NC_PLANAR_S..</td>
<td>Enabled</td>
<td></td>
<td></td>
<td>State of the planar mover, e.g. enabled.</td>
</tr>
<tr>
<td>CommandMode</td>
<td>NC_PLANAR_M..</td>
<td>RotationFree..</td>
<td></td>
<td></td>
<td>Command mode of the planar mover, e.g. onTrack.</td>
</tr>
<tr>
<td>Busy</td>
<td>MoverBusy</td>
<td></td>
<td></td>
<td></td>
<td>Busy state of the planar mover.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>HRESULT</td>
<td>16#00000000</td>
<td></td>
<td></td>
<td>Error code of the planar mover.</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td></td>
<td>REFERENCE TO...</td>
<td>%IB*</td>
<td>Move mover data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td></td>
<td>SetPos</td>
<td>MoverVector</td>
<td></td>
<td></td>
<td>Current position.</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>LREAL</td>
<td>0</td>
<td></td>
<td>X coordinate.</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>LREAL</td>
<td>0</td>
<td></td>
<td>Y coordinate.</td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>LREAL</td>
<td>0</td>
<td></td>
<td>Z coordinate.</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>LREAL</td>
<td>0</td>
<td></td>
<td>A coordinate.</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>LREAL</td>
<td>0</td>
<td></td>
<td>B coordinate.</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>LREAL</td>
<td>19,99999999..</td>
<td></td>
<td>C coordinate.</td>
</tr>
<tr>
<td></td>
<td>SetVel</td>
<td>MoverVector</td>
<td></td>
<td></td>
<td>Current velocity.</td>
</tr>
<tr>
<td></td>
<td>SetAcc</td>
<td>MoverVector</td>
<td></td>
<td></td>
<td>Current acceleration.</td>
</tr>
<tr>
<td></td>
<td>DcTimeStmp</td>
<td>UDINT</td>
<td>6915035679..</td>
<td></td>
<td>Current time stamp.</td>
</tr>
<tr>
<td></td>
<td>PhysicalAreaId</td>
<td>UDINT</td>
<td>0</td>
<td></td>
<td>Current physical area id.</td>
</tr>
<tr>
<td></td>
<td>ACT</td>
<td>REFERENCE TO...</td>
<td></td>
<td>%IB*</td>
<td>Move param data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td></td>
<td>COORDMODE</td>
<td>REFERENCE TO...</td>
<td></td>
<td>%IB*</td>
<td>Mover coordinate mode information that is transferred from the Planar Mover to the Planar Mover.</td>
</tr>
<tr>
<td></td>
<td>SETONTRACK</td>
<td>REFERENCE TO...</td>
<td></td>
<td>%IB*</td>
<td>Mover busy information that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>BOOL</td>
<td>FALSE</td>
<td></td>
<td>Flag indicating a Planar Mover error.</td>
</tr>
<tr>
<td></td>
<td>ErrorId</td>
<td>UDINT</td>
<td>0</td>
<td></td>
<td>Error id indicating the Planar Mover error type.</td>
</tr>
<tr>
<td></td>
<td>state</td>
<td>UDINT</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>target_position_c</td>
<td>LREAL</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.1.7 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", 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 Dynamic constraints.

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, curves 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 on which the Planar mover performs its synchronization movement 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 in
their meaning to the first four options of the GearInPosOnTrack command. 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 [85] to know over which Planar tracks it will
perform its movement. 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 on which the MasterSyncPosition is commanded 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 on which the MasterSyncPosition lies (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 on which the
MasterSyncPosition is located, 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.

5.2 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.

5.2.1 Configuration

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

1. Select MOTION > Add New Item...

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 tracks and confirm with OK.

The Planar track is now created and can be parameterized.

Open detailed description

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

Meaning of the individual tabs

Object: General information (name, type, ID and so on) is shown here.
Parameter (Init): Specifies initial parameters that the user can change in order to affect the behavior of the track.

The initial parameters are initially 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 have exactly the same effect as the corresponding PLC commands.

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

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

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

Data Area: Shows the memory area via which the track communicates with the PLC track.
5.2.2 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:

This is no switch.

This is no switch.
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 + 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.

If the last example is modified so that Mover 2 is in motion, a different picture may result: The earliest possible resting position of Mover 2 is less than Gap + Collision Range from the switch, therefore the additional collision avoidance would be active. If Mover 2 wants to drive on Track 3, Mover 1 would have to drive out of the way beforehand to avoid blocking Mover 2.

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).

5.2.3 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 [14].
- Create two Planar movers.
- Put "Parameter (Init)" into simulation mode (TRUE). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
- Change the start position of the second mover to x = 240.

Creating a Planar track

- Add the Planar track via Groups > Add New Item..., see Configuration [29].

Creating a PLC

- See preliminary steps under Creating a PLC [17].
- Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via MAIN.

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

7. 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.

```
Planar Motion components

```plaintext
    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

8. 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.

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

   ![PLC Untitled1 Project]

   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.

Activating and starting the project

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

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

12. Log in the PLC via the button in the menu bar.
13. 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.

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

Using this guide, you will 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 [14].

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 [29].

Creating a PLC

✓ See preliminary steps under Creating a PLC [17].

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

6. 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

7. 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.

```
Planar Motion components

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

```c
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.

9. 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.

Activating and starting the project
10. Activate the configuration via the button in the menu bar.
11. Set the TwinCAT system to the "Run" state via the button.
12. Log in the PLC via the button in the menu bar.
13. 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.

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

Using this guide, you will 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 [14].
- Create two Planar movers.
- Put "Parameter (Init)" into simulation mode (TRUE). The parameter is hidden and only becomes visible if the "Show Hidden Parameters" checkbox is activated.
- Change the start position of the second mover to x = 240.

#### Creating a Planar track

- Add the Planar track via Groups > Add New Item..., see Configuration [29].

#### Creating a PLC

- See preliminary steps under Creating a PLC [17].
- Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via MAIN.
These represent movers and tracks in the MC Configuration.

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

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

7. 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.

```plaintext
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

8. 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.

9. 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.

Activating and starting the project

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

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

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

13. 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.

5.2.6 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" [35]. Instead, the Planar mover remains coupled to an axis until a subsequent command terminates this coupling, or an error occurs.

After sending the GearInPosOnTrack 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 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 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 [14].
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 [29].
   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.
Planar Motion components

1. Select **MOTION > Add New Item…**

![Solution Explorer](image)

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

![Insert Motion Configuration](image)

- 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**.

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

---

**Creating a PLC**

- For this PLC project, you must also add "Tc2_MC2" to control the master axis; see **Inserting libraries** [90].

- See preliminary steps under **Creating a PLC** [17].
6. Create the following variables.

```plaintext
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
```

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

8. Link the Planar mover, Planar track (see Example "Joining and moving a Planar mover on the track" [35]) and the axis, as described in the next section.

**Linking an axis**

9. Double-click **Axis 1**

   ![Image of Solution Explorer with Axis 1 selected]

   in the Solution Explorer.

10. Switch to the **Settings** tab.
11. Click **Link to PLC...** in the dialog that follows select the entry **MAIN.axis** and confirm with **OK**.

![Image of the dialog box with the option to select a row](image)

**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") 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):

```plaintext
CASE state OF
  0:
    pos1.SetValuesXYC(100, 100, 0);
    pos2.SetValuesXYC(860, 100, 0);
    track.AppendLine(0, pos1, pos2);
    state := state + 1;
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
      END_IF
  1:
    mover.Enable(0);
    state := state + 1;
    IF mover.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := state + 1;
      END_IF
  2:
    mover.JoinTrack(0, track, 0, 0);
    state := state + 1;
    IF mover.MCTOPLC.STD.CommandMode = MC_PLANAR_MOVER_COMMAND_MODE.OnTrack THEN
      state := state + 1;
      END_IF
  3:
    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
  4:
    move_axis(Axis := axis,
      Position := 600,
      Velocity := 30,
      Acceleration := 100,
      Deceleration := 100,
      Jerk := 100,
      Execute := TRUE);
    state := state + 1;
    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

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

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

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

15. 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).

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mover</td>
<td>MC_PlanarMover</td>
<td></td>
</tr>
<tr>
<td>PLCTOMC</td>
<td>CDT_PLCTOMC_PLA</td>
<td></td>
</tr>
<tr>
<td>MCTOPLC</td>
<td>CDT_MCTOPLC_PLA</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>ACT</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>COORDMODE</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>SETONTRACK</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>SetPos</td>
<td>LREAL</td>
<td>274.1483212748</td>
</tr>
<tr>
<td>SetVelo</td>
<td>LREAL</td>
<td>23.99999999999999</td>
</tr>
<tr>
<td>SetAcc</td>
<td>LREAL</td>
<td>0</td>
</tr>
<tr>
<td>SetJerk</td>
<td>LREAL</td>
<td>0</td>
</tr>
<tr>
<td>TrackOID</td>
<td>OTCID</td>
<td>16#05120010</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>FALSE</td>
</tr>
<tr>
<td>ErrorId</td>
<td>UDINT</td>
<td>0</td>
</tr>
</tbody>
</table>

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 scenario is potentially returned to the PLC by the MC, but is not accepted by the above PLC code in this case. A feedback 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 this case it is possible for the slave to synchronize with its master much sooner.

A synchronization movement over a sequence of consecutive tracks is possible by using a Planar TrackTrail 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.
5.2.7 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" [44]. This example is short and builds on the above example.

Creating a Planar mover

✓ See Configuration [14].
  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 [29].

   The Solution Explorer has the following entries:

   ![Solution Explorer Screenshot]

Creating a PLC

✓ See preliminary steps under Creating a PLC [17].
  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.

```plaintext
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 "tracks".

4. Link the Planar movers and the Planar tracks (see Example "Joining and moving a Planar mover on the track" [35]).

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) is sent to the Slave Planar Mover (State 13). After the END_CASE statement, the Planar objects are updated cyclically.

```plaintext
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);
```
Planar Motion components

```plaintext
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.MCTOPLCSTD.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.MCTOPLCSTD.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, slave_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

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

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

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

8. 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" [44], 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 [84] 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 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 command is sent.

5.2.8 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.

Creating a PLC

1. See preliminary steps under Creating a PLC.
2. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via MAIN.

These represent movers and tracks in the MC Configuration.

3. Create four tracks as shown below, plus a state variable for a state machine and two auxiliary positions for the tracks.
4. 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**

5. 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.

6. To build, use the path PLC > Untitled1 > Untitled1 Project > Build.
The tracks must each be linked separately via the following dialog boxes.

Activating and starting the project

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

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

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

10. 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).

The length of each track is in the online parameters of the TCom objects in the MC Project.
5.2.9 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`, which is described in more detail in Example: "Synchronizing a Planar mover on a track with another Planar mover" \[ pag 51]. Creating and building a network of Planar tracks is explained in more detail in the Example "Connecting Planar tracks to a network" \[ pag 54]. This example is short and builds on the above examples.

Creating a Planar mover

- See Configuration \[ pag 14].
  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 \[ pag 29].
   - The Solution Explorer has the following entries:

   ![Solution Explorer](image)

Creating a PLC

- See preliminary steps under Creating a PLC \[ pag 17].
  4. Create the desired number of movers ("MC_PlanarMover") and tracks ("MC_PlanarTrack") via MAIN.
These represent movers and tracks in the MC Configuration.

5. Create the following variables.

```plaintext
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
```

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

7. Link the Planar movers and the Planar tracks (see Example "Joining and moving a Planar mover on the track" [35]).

**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 command is sent to the Slave Planar Mover. As usual, the Planar objects are updated cyclically after the END_CASE statement.

```plaintext
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;
```
Planar Motion components

```c
  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.MCPLC_STATE.Enabled THEN
    state := state + 1;
  END_IF
16:
  slave_mover.JoinTrack(0, track_in, 0, 0);
  state := state + 1;
17:
  IF slave_mover.MCPLC_STATE.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();
nslave_mover.Update();
track_in.Update();
track_out1.Update();
track_out2.Update();
move_feedback.Update();
```

Activating and starting the project

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

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

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

11. 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:
Planar Motion components

5.3 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.

5.3.1 Configuration

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

1. Select MOTION > Add New Item…
2. In the following dialog box, select **MC Configuration** and confirm with **OK**.

![Insert Motion Configuration dialog box]

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**.
Insert TcCom Object

Search: Name: [Group1 (Planar Group)] OK Cancel

Type: Beckhoff Automation GmbH
  • Motion Control
    • Internal
      • NC Group Coordinated Motion [Module]
      • CA Group [Module]
      • Planar Track [Module]
      • Planar Group [Module]
      • Planar Environment [Module]

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

Object Id: 005120020
Object Name: Group2 (Planar Group)
Type Name: Planar Group
GUID: C2D951C9-E4FA-405C-8B5E-58B3EE9EB676
Class Id: 050300C9-0000-0000-F000-000000000054
Class Factory: TcNc3
TNI/TMC: C:\\wnCAT3.1\Config\Modules\TcNc3.tmc
Parent Id: 005100010
Int Sequence: PS0

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.

Data Area: Shows the memory area via which the group communicates with the PLC track.
5.3.2 Example: "Creating and moving Planar movers with group"

Using this guide, you will to 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 [14].
  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

- Add the Planar group via Groups > Add New Item..., see Configuration [62].

Creating a PLC

- See preliminary steps under Creating a PLC [17].
  5. Create two movers ("MC_PlanarMover") and a Planar group "MC_PlanarGroup" via MAIN.
    
    PROGRAM MAIN
    VAR
      mover_one, mover_two : MC_PlanarMover;
      group : MC_PlanarGroup;
      state : UDINT;
      pos1, pos2 : PositionXYC;
    END_VAR

    7. 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
Sending the command

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

```c
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.

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

![PLC project structure](attachment:image.png)

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

10. Double-click Mover 1 first, then Mover 2.

![Link Mover to PLC](attachment:image.png)

- The group must be linked separately via the following dialog boxes.
Activating and starting the project

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

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

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

14. 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 1 has moved to x = 0 and y = 240. Mover 2 has not moved to the origin because Mover 1 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 [14].
5.4 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.

5.4.1 Configuration

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

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

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

   ![Insert Motion Configuration dialog box](image)

   - 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**.
The Planar environment is now created and can be parameterized.

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

Meaning of the individual tabs

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

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

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

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

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

### 5.4.2 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 [69].

**Creating a PLC**
- See preliminary steps Creating a PLC [17].
2. Create an "MC_PlanarEnvironment" via MAIN.
This represents the environment in the MC configuration.

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

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

4. 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

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

   ![Build Project](image)

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

   ![Project Link](image)

7. **Activating and starting the project**

   7. Activate the configuration via the button in the menu bar 🏘️.
8. Set the TwinCAT system to the "Run" state via the button.

9. Log in the PLC via the button in the menu bar.
10. 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).

5.5 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 [14].

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 and Planar group

4. Add the Planar track via Groups > Add New Item..., see Configuration [29].
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 [17].

6. Create two movers ("MC_PlanarMover"), an "MC_PlanarTrack" and an "MC_PlanarGroup" 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.

```plaintext
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 1 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 2 is blocking the track (collision error).

```plaintext
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;
    IF track.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled
    AND group.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
      END_IF
  1:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 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;
    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;
    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;
```
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:

```plaintext
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.

11. 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

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

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

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

15. 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 2 has been moved out of the way, to move Mover 1 on the track.

5.6 Planar Feedback

The MC Planar Feedback 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.

5.6.1 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 [14].
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 [17].
3. Create a mover ("MC_PlanarMover") and a Planar Feedback ("MC_PlanarFeedback") via MAIN as follows.

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

CASE state OF
  0:
    mover.Enable(feedback);
    state := 1;
  1:
    IF feedback.Done THEN
```

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:

```plaintext
  IF feedback.Done THEN
```

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

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

```plaintext
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.

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

![Image of PLC Project Settings]

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

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

7. Set the TwinCAT system to the "Run" state via the button.
8. Log in the PLC via the button in the menu bar.
9. 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.

5.6.2 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 [14].
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 [69].

Creating a Planar group

4. Create a Planar group, see Configuration [62].
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.

1. Add the libraries Tc3_Physics and Tc3_Mc3PlanarMotion to the PLC project; see Inserting libraries.
2. Create an "MC_PlanarMover" and an "MC_PlanarEnvironment" via MAIN.

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

```plaintext
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.

8. 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.
Subsequently, the Planar environment can be linked via the following dialog boxes in the "MC Project".

The group is linked in the same way.

Activating and starting the project

9. Activate the configuration via the button in the menu bar.
10. Set the TwinCAT system to the "Run" state via the button.

11. Log in the PLC via the button in the menu bar.
12. 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.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
<th>Prepared value</th>
<th>Address</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>mover</td>
<td>MC_PlanarMover</td>
<td>CDT_PLC_MVC</td>
<td>%Q&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Mover data that is transferred from the PlanarFeedback.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDT_MVC_PLANAR</td>
<td>%Q&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Mover data that is transferred from the PlanarFeedback.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>%Q&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Mover standard data is transferred from...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REFERENCE TO CDT...</td>
<td>%Q&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Mover setpoint data is transferred from...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MvP</td>
<td>%Q&lt;sup&gt;5&lt;/sup&gt;</td>
<td>Mover position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>REAL</td>
<td>0</td>
<td>X coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>REAL</td>
<td>0</td>
<td>Y coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>REAL</td>
<td>0</td>
<td>Z coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>REAL</td>
<td>0</td>
<td>A coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>REAL</td>
<td>0</td>
<td>B coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>REAL</td>
<td>0</td>
<td>C coordinate.</td>
<td></td>
</tr>
<tr>
<td>SetPos</td>
<td>MvP</td>
<td>%Q&lt;sup&gt;6&lt;/sup&gt;</td>
<td>Mover position.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>REAL</td>
<td>0</td>
<td>X coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>REAL</td>
<td>0</td>
<td>Y coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>z</td>
<td>REAL</td>
<td>0</td>
<td>Z coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>REAL</td>
<td>0</td>
<td>A coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>REAL</td>
<td>0</td>
<td>B coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>REAL</td>
<td>0</td>
<td>C coordinate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetVelo</td>
<td>MvP</td>
<td>%Q&lt;sup&gt;7&lt;/sup&gt;</td>
<td>Current velocity.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetAcc</td>
<td>MvP</td>
<td>%Q&lt;sup&gt;8&lt;/sup&gt;</td>
<td>Current acceleration.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DrTimeStmp</td>
<td>UINT</td>
<td>6431588341751</td>
<td>Current time stamp.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PhysicalAreaID</td>
<td>UINT</td>
<td>0</td>
<td>Current physical area id.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACT</td>
<td>CDT_MVC_PLANAR</td>
<td>%Q&lt;sup&gt;9&lt;/sup&gt;</td>
<td>Mover setpoint data is transferred from...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COORD_MODE</td>
<td>CDT_MVC_PLANAR</td>
<td>%Q&lt;sup&gt;10&lt;/sup&gt;</td>
<td>Mover coordinate mode information that is transferred from...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MASK</td>
<td>CDT_MVC_PLANAR</td>
<td>%Q&lt;sup&gt;11&lt;/sup&gt;</td>
<td>Mover busy information that is transferred from...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inSync</td>
<td>BOOL</td>
<td>FALSE</td>
<td>Flag indicating a PlanarMover error.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>errorId</td>
<td>UINT</td>
<td>0</td>
<td>Error id indicating the PlanarMover error type.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>environment</td>
<td>MC_PlanarEnvironment</td>
<td></td>
<td>Indicates which object one would collide with.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>group</td>
<td>MC_PlanarGroup</td>
<td></td>
<td>Object group.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>feedback</td>
<td>MC_PlanarFeedback</td>
<td></td>
<td>Feedback object.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>objectInfo</td>
<td>PlaneObjectInfo</td>
<td></td>
<td>Indicates which object one would collide with.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ObjectType</td>
<td>EPLANAROBJECTTYPE</td>
<td>Environment</td>
<td>Object type.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id</td>
<td>UINT</td>
<td>8356797</td>
<td>Object id.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>BOOL</td>
<td>FALSE</td>
<td>Indicates an active command, i.e., command was...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Busy</td>
<td>BOOL</td>
<td>FALSE</td>
<td>Indicates a busy command, i.e., command is busy...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Done</td>
<td>BOOL</td>
<td>FALSE</td>
<td>Indicates the command done, i.e., execution...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aborted</td>
<td>BOOL</td>
<td>FALSE</td>
<td>Indicates the command aborted, i.e., execution...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>BOOL</td>
<td>TRUE</td>
<td>Indicates the command has an error.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>errorId</td>
<td>UINT</td>
<td>23510</td>
<td>Error id indicating the error.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>state</td>
<td>UINT</td>
<td>6</td>
<td>Indicates the error id of the command error.</td>
<td></td>
</tr>
</tbody>
</table>

### 5.6.3 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 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". 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" [51]. It has an additional output inSync, which indicates whether the executing mover is synchronous with the Master Planar Mover.

5.7 Planar TrackTrail

The MC_PlanarTrackTrail is an object that defines a path of consecutive Planar tracks in a network. Unlike the individual Planar tracks from which the Planar TrackTrail is built, the Planar TrackTrail has no fixed equivalent in a TCOM object on the MC side, but is declared solely in the PLC, similar to a Planar Feedback [77].

A PlanarTrackTrail can be used to define a path of Planar tracks over which a synchronization movement of a Slave Planar Mover Example "Synchronizing a Planar mover on a track with one axis" [44] or Example: "Synchronizing a Planar mover on a track with another Planar mover" [51] should take place (if this path consists of more than the current Planar track of the Slave Planar Mover).

The Planar TrackTrail provides methods for adding a Planar track at its end and emptying its configuration. These methods modify only the Planar TrackTrail and, in particular, 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 impossible for a Planar track to be added more than once.
5.7.1 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" [44], 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:

Customizing the PLC project

1. Add the libraries Tc3_Mc3PlanarMotion, Tc3_Physics and Tc2_MC2 to the PLC project; see Inserting libraries [90].

2. Declare the following variables in MAIN:

```plaintext
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" [p. 18]") and the axis (see example "Example: "Synchronizing a Planar mover on a track with one axis" [p. 44]).

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" [p. 44].

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" [p. 54], consisting of an L-configuration with a loop piece:

```plaintext
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);
```
Planar Motion components

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();
trail1.Update();
trail2.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

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

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

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

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

Observe the process in the online view

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mover</td>
<td>MC_PlanarMover</td>
<td></td>
</tr>
<tr>
<td>PLCTOMIC</td>
<td>CDT_PLC reality</td>
<td></td>
</tr>
<tr>
<td>MCTOPLC</td>
<td>CDT_MCTOPLC</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>ACT</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>COORDMODE</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>SETONTRACK</td>
<td>REFERENCE TO CDT</td>
<td></td>
</tr>
<tr>
<td>SetPos</td>
<td>LREAL</td>
<td>1.908683232748...</td>
</tr>
<tr>
<td>SetVel</td>
<td>LREAL</td>
<td>29.99999999998...</td>
</tr>
<tr>
<td>SetAcc</td>
<td>LREAL</td>
<td>0</td>
</tr>
<tr>
<td>SetJerk</td>
<td>LREAL</td>
<td>0</td>
</tr>
<tr>
<td>TrackOID</td>
<td>OTCID</td>
<td>16#05123010</td>
</tr>
</tbody>
</table>

10. You will then see it switch to the subsequent Planar track (note the TrackOIDs):
Finally, you can see how it comes to a standstill on the second Planar track:

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 \[77\] (which are not used in this example), is necessary, and a corresponding method is not available.
6  PLC Libraries

6.1  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.
6.2 Tc3_Mc3PlanarMotion API

6.2.1 Data Types

6.2.1.1 Enums

6.2.1.1.1 EPlanarObjectType

Identifies a planar object type.

Syntax

Definition:

```plaintext
TYPE EPlanarObjectType :
{
    Invalid := 0,
    None := 301,
    Mover := 302,
    Track := 303,
    Environment := 304
}UINT;
END_TYPE
```

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>Indicates invalid information, e.g. no connection or component not yet ready.</td>
</tr>
<tr>
<td>None</td>
<td>No planar object.</td>
</tr>
<tr>
<td>Mover</td>
<td>Planar Mover.</td>
</tr>
<tr>
<td>Track</td>
<td>Planar Track.</td>
</tr>
<tr>
<td>Environment</td>
<td>Planar Environment.</td>
</tr>
</tbody>
</table>

6.2.1.1.2 MC_DIRECTION

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

Syntax

Definition:

```plaintext
TYPE MC_DIRECTION :
{
    mcDirectionNonModulo := 0,
    mcDirectionPositive := 1,
    mcDirectionShortestWay := 2,
    mcDirectionNegative := 3
}UINT;
END_TYPE
```

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcDirectionNonModulo</td>
<td>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.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mcDirectionPositive</td>
<td>The Planar Mover moves to the target position in a forward direction. No backward movement is allowed.</td>
</tr>
<tr>
<td>mcDirectionShortestWay</td>
<td>The Planar Mover takes the shortest way to the target position. May induce forward or backward movement.</td>
</tr>
<tr>
<td>mcDirectionNegative</td>
<td>The Planar Mover moves to the target position in a backward direction. No forward movement is allowed.</td>
</tr>
</tbody>
</table>

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).

### 6.2.1.1.3 MC_SYNC_DIRECTIONS

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

**Syntax**

**Definition:**

```plaintext
TYPE MC_SYNC_DIRECTIONS :
{
    Invalid  := 0,
    Positive := 1,
    Negative := 2,
    Both     := 3
}UINT;
END_TYPE
```

**Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>Indicates invalid information, e.g. no connection or component not yet ready.</td>
</tr>
<tr>
<td>Positive</td>
<td>Movement is allowed only in positive direction while synchronizing.</td>
</tr>
<tr>
<td>Negative</td>
<td>Movement is allowed only in negative direction while synchronizing.</td>
</tr>
<tr>
<td>Both</td>
<td>Movement is allowed in any direction while synchronizing.</td>
</tr>
</tbody>
</table>

### 6.2.1.2 Structs

#### 6.2.1.2.1 CDT_MCTOPLC_PLANAR_MOVER

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

**Syntax**

**Definition:**

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

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>CDT_MCTOPLC_PLANAR_MOVER_R_STD</td>
<td>Mover standard data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td>SET</td>
<td>CDT_MCTOPLC_PLANAR_MOVER_R_SET</td>
<td>Mover setpoint data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td>ACT</td>
<td>CDT_MCTOPLC_PLANAR_MOVER_R_ACT</td>
<td>Mover actpoint data that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td>COORDMODE</td>
<td>CDT_MCTOPLC_PLANAR_MOVER_RCOORDMODE</td>
<td>Mover coordinate mode information that is transferred from the Planar Mover to this function block.</td>
</tr>
<tr>
<td>SETONTRACK</td>
<td>CDT_MCTOPLC_PLANAR_MOVER_R_TRACK</td>
<td>Mover busy information that is transferred from the Planar Mover to this function block.</td>
</tr>
</tbody>
</table>

#### 6.2.1.2.2 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 : CDT_PLCTOMC_PLANAR_MOVER_STD;
END_STRUCT
END_TYPE
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>CDT_PLCTOMC_PLANAR_MOVER_R_STD</td>
<td>Mover standard data that is transferred from this function block to the Planar Mover.</td>
</tr>
</tbody>
</table>

#### 6.2.1.2.3 PlanarObjectInfo

Identifies a planar object uniquely by object id and type.

**Syntax**

Definition:

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

**TF5430**  
Version: 1.3
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>EPlanarObjectType</td>
<td></td>
<td>Object type.</td>
</tr>
<tr>
<td>Id</td>
<td>UDINT</td>
<td></td>
<td>Object id.</td>
</tr>
</tbody>
</table>

6.2.1.2.4 ST_AdoptTrackOrientationOptions

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

Syntax

Definition:

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

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalTurns</td>
<td>UDINT</td>
<td>0</td>
<td>Addition turns to move in modulo movement (positive or negative).</td>
</tr>
<tr>
<td>direction</td>
<td>MC_DIRECTION</td>
<td>MC_DIRECTION.mcDirectionShortestWay</td>
<td>Direction in which the target is approached.</td>
</tr>
</tbody>
</table>

6.2.1.2.5 ST_ExternalSetpointGenerationOptions

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

Syntax

Definition:

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

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>MC_EXTERNAL_SET_POSITION_MODE</td>
<td>MC_EXTERNAL_SET_POSITION_MODE.Absolute</td>
<td>Mode can be relative or absolute, relative can be used parallel to all other commands, absolute only alone.</td>
</tr>
</tbody>
</table>

6.2.1.2.6 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gap</td>
<td>LREAL</td>
<td>200.0</td>
<td>Minimal distance to next Planar Mover on track.</td>
</tr>
<tr>
<td>inSyncToleranceDistance</td>
<td>LREAL</td>
<td>0.0</td>
<td>Tolerance in absolute value of position difference to master axis for inSync flag.</td>
</tr>
<tr>
<td>directionSlaveSyncPosition</td>
<td>MC_DIRECTION</td>
<td>MC_DIRECTION.mcDirectionNonModulo</td>
<td>Direction in which the slave sync position is approached.</td>
</tr>
<tr>
<td>moduloToleranceSlaveSyncPosition</td>
<td>LREAL</td>
<td>0.0</td>
<td>Tolerance &quot;window&quot; for slave sync position.</td>
</tr>
<tr>
<td>allowedSlaveSyncDirections</td>
<td>MC_SYNC_DIRECTIONS</td>
<td>MC_SYNC_DIRECTIONS .Positive</td>
<td>Directions in which the slave is allowed to move while in synchronizing phase.</td>
</tr>
</tbody>
</table>

6.2.1.2.7 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gap</td>
<td>LREAL</td>
<td>200.0</td>
<td>Minimal distance to next Planar Mover on track.</td>
</tr>
<tr>
<td>inSyncToleranceDistance</td>
<td>LREAL</td>
<td>0.0</td>
<td>Tolerance in absolute value of position difference to master axis for inSync flag.</td>
</tr>
<tr>
<td>directionSlaveSyncPosition</td>
<td>MC_DIRECTION</td>
<td>MC_DIRECTION.mcDirectionNonModulo</td>
<td>Direction in which the slave sync position is approached.</td>
</tr>
</tbody>
</table>

TF5430 Version: 1.3 95
### 6.2.1.2.8 ST_JoinTrackOptions

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

**Syntax**

**Definition:**

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

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useOrientation</td>
<td>BOOL</td>
<td>TRUE</td>
<td>If true, the target orientation is also reached at the end of the movement.</td>
</tr>
</tbody>
</table>

### 6.2.1.2.9 ST_LeaveTrackOptions

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

**Syntax**

**Definition:**


TYPE ST_LeaveTrackOptions :
  STRUCT
    useOrientation : BOOL;
  END_STRUCT
END_TYPE

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useOrientation</td>
<td>BOOL</td>
<td>TRUE</td>
<td>If true, the target orientation is also reached at the end of the movement.</td>
</tr>
</tbody>
</table>

6.2.1.2.10 ST_MoveCOptions

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

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalTurns</td>
<td>UDINT</td>
<td>0</td>
<td>Addition turns to move in modulo movement (positive or negative).</td>
</tr>
<tr>
<td>direction</td>
<td>MC_DIRECTION [91]</td>
<td></td>
<td>Direction in which the target is approached.</td>
</tr>
</tbody>
</table>

6.2.1.2.11 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gap</td>
<td>LREAL</td>
<td>200.0</td>
<td>Minimal distance to next Planar Mover on track.</td>
</tr>
<tr>
<td>direction</td>
<td>MC_DIRECTION [91]</td>
<td></td>
<td>Direction in which the target is approached.</td>
</tr>
</tbody>
</table>
### 6.2.1.2.12 ST_MoveToPositionOptions

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

#### Syntax

**Definition:**

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

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useOrientation</td>
<td>BOOL</td>
<td>TRUE</td>
<td>If true, the target orientation is also reached at the end of the movement.</td>
</tr>
</tbody>
</table>

### 6.2.2 Function Blocks

#### 6.2.2.1 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>Clears the Planar Environment (stators and boundary).</td>
</tr>
<tr>
<td>AddStator</td>
<td>Adds a stator to the Planar Environment.</td>
</tr>
<tr>
<td>CreateBoundary</td>
<td>Creates a boundary for the Planar Environment based on the previously added stator information or hardware information.</td>
</tr>
<tr>
<td>Update</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
<tr>
<td>AddToGroup</td>
<td>Adds the Planar Environment to the given Planar Group.</td>
</tr>
<tr>
<td>RemoveFromGroup</td>
<td>Removes the Planar Environment from its current Planar Group, i.e. disables collision checks.</td>
</tr>
<tr>
<td>GetPlanarObjectInfo</td>
<td>Returns environment object info (type: environment, id: OID of nc environment).</td>
</tr>
</tbody>
</table>

### Required License

TC3 Planar Motion Base
System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.12</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.1.1 Clear

Clear

commandFeedback : MC_PlanarFeedback

Clears the Planar Environment (stators and boundary).

Syntax

Definition:

METHOD Clear
VAR_INPUT
   commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.1.2 AddStator

AddStator

commandFeedback : MC_PlanarFeedback

lowerX : LREAL

lowerY : LREAL

Adds a stator to the Planar Environment.

Syntax

Definition:

METHOD AddStator
VAR_INPUT
   commandFeedback : MC_PlanarFeedback;
   lowerX          : LREAL;
   lowerY          : LREAL;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>lowerX</td>
<td>LREAL</td>
<td>The lower x position of the stator.</td>
</tr>
<tr>
<td>lowerY</td>
<td>LREAL</td>
<td>The lower y position of the stator.</td>
</tr>
</tbody>
</table>
6.2.2.1.3 CreateBoundary

Create a boundary for the Planar Environment based on the previously added stator information or hardware information.

Syntax

Definition:

```
METHOD CreateBoundary
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR
```

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

6.2.2.1.4 Update

Updates internal state of the object, must be triggered each cycle.

Syntax

Definition:

```
METHOD Update
```

6.2.2.1.5 AddToGroup

Adds the Planar Environment to the given Planar Group.

Syntax

Definition:

```
METHOD AddToGroup
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
  group : MC_PlanarGroup;
END_VAR
```
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

## In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>MC_PlanarGroup</td>
<td>The Planar Group that the mover joins.</td>
</tr>
</tbody>
</table>

### 6.2.2.1.6 RemoveFromGroup

RemoveFromGroup

```plaintext
commandFeedback : MC_PlanarFeedback
```

Removes the Planar Environment from its current Planar Group, i.e. disables collision checks.

### Syntax

**Definition:**

```plaintext
METHOD RemoveFromGroup
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
END_VAR
```

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

### 6.2.2.1.7 GetPlanarObjectInfo

GetPlanarObjectInfo

```plaintext
PlanarObjectInfo
```

Returns environment object info (type: environment, id: OID of nc environment).

### Syntax

**Definition:**

```plaintext
METHOD GetPlanarObjectInfo : PlanarObjectInfo
```

### Return value

PlanarObjectInfo [ 93 ]
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;
  Done   : BOOL;
  Aborted: BOOL;
  Error  : BOOL;
  ErrorId: UDINT;
  objectInfo : PlanarObjectInfo;

END_VAR

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>BOOL</td>
<td>Indicates an active command, i.e. command was accepted and is being executed.</td>
</tr>
<tr>
<td>Busy</td>
<td>BOOL</td>
<td>Indicates a busy command, i.e. command is being processed, waiting for execution, or already executing (= also active).</td>
</tr>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>Indicates the command is done, i.e. execution of the command finished successfully.</td>
</tr>
<tr>
<td>Aborted</td>
<td>BOOL</td>
<td>Indicates the command is aborted, i.e. execution of the command finished due the start of other commands.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>Indicates the command has an error.</td>
</tr>
<tr>
<td>ErrorId</td>
<td>UDINT</td>
<td>Indicates the error id of the command error.</td>
</tr>
<tr>
<td>objectInfo</td>
<td>PlanarObjectInfo</td>
<td>Indicates which object one would collide with.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Updates internal state of the object.</td>
</tr>
</tbody>
</table>

### Required License

TC3 Planar Motion Base

### System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.12</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2.2.2.1 Update

Updates internal state of the object.

**Syntax**

Definition:

```plaintext
METHOD Update
```

6.2.2.3 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.17</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.4 MC_PlanarFeedbackGearInPosOnTrack

Displays specific command status information for an associated GearInPosOnTrack command, given back by the MC Project.

**Syntax**

Definition:

```plaintext
FUNCTION_BLOCK MC_PlanarFeedbackGearInPosOnTrack
VAR_OUTPUT
    Active : BOOL;
    Busy  : BOOL;
    Done  : BOOL;
    Aborted : BOOL;
    Error  : BOOL;
    ErrorId : UDINT;
    objectInfo : PlanarObjectInfo;
    inSync : BOOL;
END_VAR
```
## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>BOOL</td>
<td>Indicates an active command, i.e. command was accepted and is being executed.</td>
</tr>
<tr>
<td>Busy</td>
<td>BOOL</td>
<td>Indicates a busy command, i.e. command is being processed, waiting for execution, or already executing (= also active).</td>
</tr>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>Indicates the command is done, i.e. execution of the command finished successfully.</td>
</tr>
<tr>
<td>Aborted</td>
<td>BOOL</td>
<td>Indicates the command is aborted, i.e. execution of the command finished due the start of other commands.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>Indicates the command has an error.</td>
</tr>
<tr>
<td>ErrorId</td>
<td>UDINT</td>
<td>Indicates the error id of the command error.</td>
</tr>
<tr>
<td>objectInfo</td>
<td>PlanarObjectInfo</td>
<td>Indicates which object one would collide with.</td>
</tr>
<tr>
<td>inSync</td>
<td>BOOL</td>
<td>Indicates whether the mover is currently in sync with the master (within tolerance specified in the command options).</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Updates internal state of the object.</td>
</tr>
</tbody>
</table>

## Required License

TC3 Planar Motion Base

## System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.17</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.2.2.4.1 Update

`Update` updates the internal state of the object.

#### Syntax

**Definition:**

``` METHOD Update ```
6.2.2.5  MC_PlanarFeedbackGearInPosOnTrackWithMasterMover

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;
    Done        : BOOL;
    Aborted     : BOOL;
    Error       : BOOL;
    ErrorId     : UDINT;
    objectInfo  : PlanarObjectInfo;
    inSync      : BOOL;
END_VAR

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>BOOL</td>
<td>Indicates an active command, i.e. command was accepted and is being executed.</td>
</tr>
<tr>
<td>Busy</td>
<td>BOOL</td>
<td>Indicates a busy command, i.e. command is being processed, waiting for execution, or already executing (= also active).</td>
</tr>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>Indicates the command is done, i.e. execution of the command finished successfully.</td>
</tr>
<tr>
<td>Aborted</td>
<td>BOOL</td>
<td>Indicates the command is aborted, i.e. execution of the command finished due the start of other commands.</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>Indicates the command has an error.</td>
</tr>
<tr>
<td>ErrorId</td>
<td>UDINT</td>
<td>Indicates the error id of the command error.</td>
</tr>
<tr>
<td>objectInfo</td>
<td>PlanarObjectInfo</td>
<td>Indicates which object one would collide with.</td>
</tr>
<tr>
<td>inSync</td>
<td>BOOL</td>
<td>Indicates whether the mover is currently in sync with the master (within tolerance specified in the command options).</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Updates internal state of the object.</td>
</tr>
</tbody>
</table>

**Required License**

TC3 Planar Motion Base
System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.17</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.5.1 Update

**Update**

Updates internal state of the object.

**Syntax**

**Definition:**

```
METHOD Update
```

6.2.2.6 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.17</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.7 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable [107]</td>
<td>Starts enabling the Planar Group.</td>
</tr>
<tr>
<td>Disable [107]</td>
<td>Starts disabling the Planar Group.</td>
</tr>
<tr>
<td>Reset [107]</td>
<td>Starts resetting the Planar Group.</td>
</tr>
<tr>
<td>Update [108]</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
</tbody>
</table>

**Required License**

TC3 Planar Motion Base
System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.12</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.7.1 Enable

Enable commandFeedback MC_PlanarFeedback

Starts enabling the Planar Group.

Syntax

Definition:

METHOD Enable
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.7.2 Disable

Disable commandFeedback MC_PlanarFeedback

Starts disabling the Planar Group.

Syntax

Definition:

METHOD Disable
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.7.3 Reset

Reset commandFeedback MC_PlanarFeedback
Starts resetting the Planar Group.

**Syntax**

**Definition:**

```plaintext
METHOD Reset
VAR_INPUT
  _commandFeedback : MC_PlanarFeedback;
END_VAR
```

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

#### 6.2.2.7.4 Update

**Update**

Updates internal state of the object, must be triggered each cycle.

**Syntax**

**Definition:**

```plaintext
METHOD Update
```

### 6.2.2.8 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveToPosition [109]</td>
<td>Initiates a direct movement to the specified position.</td>
</tr>
<tr>
<td>JoinTrack [110]</td>
<td>Initiates a direct movement to the specified track. At the end of the movement the mover joins the track.</td>
</tr>
<tr>
<td>LeaveTrack [111]</td>
<td>Initiates a direct movement to the specified position. At the beginning of the movement the track is left.</td>
</tr>
<tr>
<td>MoveOnTrack [111]</td>
<td>Initiates a movement on the track to the specified position and returns command ID.</td>
</tr>
<tr>
<td>GearInPosOnTrack [112]</td>
<td>Initiates a GearInPos movement along a specified trail.</td>
</tr>
<tr>
<td>GearInPosOnTrackWithMasterMover [113]</td>
<td>Initiates a GearInPos movement along a specified trail, in which the master setpoints are provided by another PlanarMover.</td>
</tr>
<tr>
<td>MoveZ [114]</td>
<td>Initiates a movement for the z component.</td>
</tr>
<tr>
<td>MoveA [115]</td>
<td>Initiates a movement for the a component.</td>
</tr>
<tr>
<td>MoveB [115]</td>
<td>Initiates a movement for the b component.</td>
</tr>
<tr>
<td>MoveC [116]</td>
<td>Initiates a movement for the c component.</td>
</tr>
<tr>
<td>AdoptTrackOrientation [116]</td>
<td>Initiates a movement for the c component.</td>
</tr>
<tr>
<td>Halt [117]</td>
<td>Initiates a halt.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Enable [1.17]</td>
<td>Starts enabling the Planar Mover.</td>
</tr>
<tr>
<td>Disable [1.18]</td>
<td>Starts disabling the Planar Mover.</td>
</tr>
<tr>
<td>Reset [1.18]</td>
<td>Starts resetting the Planar Mover.</td>
</tr>
<tr>
<td>Update [1.18]</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
<tr>
<td>SetPosition [1.19]</td>
<td>Sets the position of the Planar Mover. Only possible if the Planar Mover is disabled.</td>
</tr>
<tr>
<td>StartExternalSetpointGeneratio[n [1.19]</td>
<td>Starts the external setpoint generation, the user must supply setpoints from this PLC cycle on in every PLC cycle.</td>
</tr>
<tr>
<td>StopExternalSetpointGeneration [1.20]</td>
<td>Ends the external setpoint generation, called after last SetExternalSetpoint (in the same PLC cycle).</td>
</tr>
<tr>
<td>SetExternalSetpoint [1.20]</td>
<td>Sets the external setpoint for the Planar Mover, must be called each PLC cycle during external setpoint generation.</td>
</tr>
<tr>
<td>AddToGroup [1.20]</td>
<td>Adds the Planar Mover to the given Planar Group.</td>
</tr>
<tr>
<td>RemoveFromGroup [1.12]</td>
<td>Removes the Planar Mover from its current Planar Group, i.e. disables collision checks.</td>
</tr>
</tbody>
</table>

**Required License**

TC3 Planar Motion Base

**System Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.12</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**6.2.2.8.1 MoveToPosition**

Initiates a direct movement to the specified position.

**Syntax**

**Definition:**

METHOD MoveToPosition
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  constraint : IPlcDynamicConstraint;
  options : ST_MoveToPositionOptions;
  targetPosition : PositionXYC;
END_VAR
VAR_IN_OUT
  targetPosition : PositionXYC;
END_VAR
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
<tr>
<td>options</td>
<td>ST_MoveToPositionOptions</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

## In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetPosition</td>
<td>PositionXYC</td>
<td>Target position for the movement.</td>
</tr>
</tbody>
</table>

### 6.2.2.8.2 JoinTrack

**JoinTrack**

<table>
<thead>
<tr>
<th>commandFeedback</th>
<th>MC_PlanarFeedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
</tr>
<tr>
<td>options</td>
<td>ST_JoinTrackOptions</td>
</tr>
<tr>
<td>targetTrack</td>
<td>MC_PlanarTrack</td>
</tr>
</tbody>
</table>

Initiates a direct movement to the specified track. At the end of the movement the mover joins the track.

**Syntax**

**Definition:**

```plaintext
METHOD JoinTrack
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  constraint      : IPlcDynamicConstraint;
  options         : ST_JoinTrackOptions;
END_VAR
VAR_IN_OUT
  targetTrack     : MC_PlanarTrack;
END_VAR
```

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
<tr>
<td>options</td>
<td>ST_JoinTrackOptions</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

## In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetTrack</td>
<td>MC_PlanarTrack</td>
<td>Target track for the movement.</td>
</tr>
</tbody>
</table>
6.2.2.8.3 LeaveTrack

LeaveTrack

commandFeedback : MC_PlanarFeedback
constraint : IPlcDynamicConstraint
options : 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 : MC_PlanarFeedback;
  constraint : IPlcDynamicConstraint;
  options : ST_LeaveTrackOptions;
END_VAR
VAR_IN_OUT
  targetPosition : PositionXYC;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
<tr>
<td>options</td>
<td>ST_LeaveTrackOptions</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetPosition</td>
<td>PositionXYC</td>
<td>Target position for the movement.</td>
</tr>
</tbody>
</table>

6.2.2.8.4 MoveOnTrack

MoveOnTrack

commandFeedback : MC_PlanarFeedback
targetTrack : MC_PlanarTrack
targetPositionOnTrack : LREAL
constraint : DynamicConstraint_PathXY
options : ST_MoveOnTrackOptions

Initiates a movement on the track to the specified position and returns command ID.

Syntax

Definition:
METHOD MoveOnTrack
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  targetTrack : MC_PlanarTrack;
  targetPositionOnTrack : LREAL;
  constraint : DynamicConstraint_PathXY;
  options : ST_MoveOnTrackOptions;
END_VAR
constraint : DynamicConstraint_PathXY;
options : ST_MoveOnTrackOptions;
END_VAR

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>targetTrack</td>
<td>MC_PlanarTrack [122]</td>
<td>Target track for the movement. If none is specified, this defaults to the current track.</td>
</tr>
<tr>
<td>targetPositionOnTrack</td>
<td>LREAL</td>
<td>Target position on the target track.</td>
</tr>
<tr>
<td>constraint</td>
<td>DynamicConstraint_PathXY</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
<tr>
<td>options</td>
<td>ST_MoveOnTrackOptions</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

#### 6.2.2.8.5 GearInPosOnTrack

**GearInPosOnTrack**

```plaintext
commandFeedback   MC_PlanarFeedbackGearInPosOnTrack
masterAxis        OTCID
trackTrail        MC_PlanarTrackTrail
masterSyncPosition LREAL
slaveSyncPosition LREAL
constraint        DynamicConstraint_PathXY
options          ST_GearInPosOnTrackOptions
slaveSyncPositionTrack : MC_PlanarTrack;
```

Initiates a GearInPos movement along a specified trail.

**Syntax**

**Definition:**

```plaintext
METHOD GearInPosOnTrack
VAR_INPUT
    commandFeedback   : MC_PlanarFeedbackGearInPosOnTrack;
    masterAxis        : OTCID;
    trackTrail        : MC_PlanarTrackTrail;
    masterSyncPosition : LREAL;
    slaveSyncPosition : LREAL;
    constraint        : DynamicConstraint_PathXY;
    options           : ST_GearInPosOnTrackOptions;
END_VAR
VAR_IN_OUT
    slaveSyncPositionTrack : MC_PlanarTrack;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedbackGearInPosOnTrack</td>
<td>The command specific feedback object for the command.</td>
</tr>
<tr>
<td>masterAxis</td>
<td>OTCID</td>
<td>Master axis being followed.</td>
</tr>
<tr>
<td>trackTrail</td>
<td>MC_PlanarTrackTrail</td>
<td>Track trail determining along which tracks the GearInPos movement is allowed to proceed.</td>
</tr>
</tbody>
</table>
### PLC Libraries

#### TF5430

**Version:** 1.3

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>masterSyncPosition</td>
<td>LREAL</td>
<td>Position of the master axis at which the slave is inSync.</td>
</tr>
<tr>
<td>slaveSyncPosition</td>
<td>LREAL</td>
<td>Arc length on track given by slaveSyncPositionTrackOID at which the slave is inSync. Possibly interpreted in modulo fashion, depending on options.</td>
</tr>
<tr>
<td>constraint</td>
<td>DynamicConstraint_PathXY</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
<tr>
<td>options</td>
<td>ST_GearInPosOnTrackOptions [94]</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

---

#### In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slaveSyncPositionTrack</td>
<td>MC_PlanarTrack [122]</td>
<td>Track on which the slave is inSync.</td>
</tr>
</tbody>
</table>

---

**6.2.2.8.6 GearInPosOnTrackWithMasterMover**

**Syntax**

**Definition:**

```plaintext
METHOD GearInPosOnTrackWithMasterMover
VAR_INPUT
  commandFeedback : MC_PlanarFeedbackGearInPosOnTrackWithMasterMover;
  trackTrail     : MC_PlanarTrackTrail;
  masterSyncPosition : LREAL;
  slaveSyncPosition : LREAL;
  constraint     : DynamicConstraint_PathXY;
  options        : ST_GearInPosOnTrackWithMasterMoverOptions;
END_VAR
VAR_IN_OUT
  masterMover       : MC_PlanarMover;
  masterSyncPositionTrack : MC_PlanarTrack;
  slaveSyncPositionTrack : MC_PlanarTrack;
END_VAR
```

Initiates a GearInPos movement along a specified trail, in which the master setpoints are provided by another PlanarMover.

---

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedbackGearInPosOnTrackWithMasterMover</td>
<td>The command specific feedback object for the command.</td>
</tr>
</tbody>
</table>
### PLC Libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trackTrail</td>
<td>MC_PlanarTrackTrail [130]</td>
<td>Track trail determining along which tracks the GearInPos movement is allowed to proceed.</td>
</tr>
<tr>
<td>masterSyncPosition</td>
<td>LREAL</td>
<td>Position of the master axis at which the slave is inSync.</td>
</tr>
<tr>
<td>slaveSyncPosition</td>
<td>LREAL</td>
<td>Arc length on track given by slaveSyncPositionTrackOID at which the slave is inSync. Possibly interpreted in modulo fashion, depending on options.</td>
</tr>
<tr>
<td>constraint</td>
<td>DynamicConstraint_PathXY</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
<tr>
<td>options</td>
<td>ST_GearInPosOnTrackWithMasterMoverOptions [95]</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

### In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>masterSyncPositionTrack</td>
<td>MC_PlanarTrack [122]</td>
<td>Track on which the master is inSync.</td>
</tr>
<tr>
<td>slaveSyncPositionTrack</td>
<td>MC_PlanarTrack [122]</td>
<td>Track on which the slave is inSync.</td>
</tr>
</tbody>
</table>

### 6.2.2.8.7 MoveZ

**MoveZ**

- commandFeedback : MC_PlanarFeedback
- targetPosition : LREAL
- constraint : IPlcDynamicConstraint

Initiates a movement for the z component.

### Syntax

**Definition:**

```plaintext
METHOD MoveZ
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  targetPosition : LREAL;
  constraint : IPlcDynamicConstraint;
END_VAR
```

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetPosition</td>
<td>LREAL</td>
<td>Target position for the movement.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
</tbody>
</table>
6.2.2.8.8 MoveA

Initiates a movement for the a component.

Syntax

Definition:

```
METHOD MoveA
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    targetPosition  : LREAL;
    constraint      : IPlcDynamicConstraint;
END_VAR
```

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>targetPosition</td>
<td>LREAL</td>
<td>Target position for the movement.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
</tbody>
</table>

6.2.2.8.9 MoveB

Initiates a movement for the b component.

Syntax

Definition:

```
METHOD MoveB
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    targetPosition  : LREAL;
    constraint      : IPlcDynamicConstraint;
END_VAR
```

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>targetPosition</td>
<td>LREAL</td>
<td>Target position for the movement.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
</tbody>
</table>
6.2.2.8.10 MoveC

**MoveC**

- **commandFeedback**: `MC_PlanarFeedback`
- **targetPosition**: `LREAL`
- **constraint**: `IPlcDynamicConstraint`
- **options**: `ST_MoveCOptions`

Initiates a movement for the c component.

**Syntax**

Definition:

```plaintext
METHOD MoveC
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    targetPosition   : LREAL;
    constraint       : IPlcDynamicConstraint;
    options          : ST_MoveCOptions;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td><code>MC_PlanarFeedback</code></td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>targetPosition</td>
<td><code>LREAL</code></td>
<td>Target position for the movement.</td>
</tr>
<tr>
<td>constraint</td>
<td><code>IPlcDynamicConstraint</code></td>
<td>Dynamic constraints for this movement.</td>
</tr>
<tr>
<td>options</td>
<td><code>ST_MoveCOptions</code></td>
<td>Options for the rotation.</td>
</tr>
</tbody>
</table>

6.2.2.8.11 AdoptTrackOrientation

**AdoptTrackOrientation**

- **commandFeedback**: `MC_PlanarFeedback`
- **constraint**: `IPlcDynamicConstraint`
- **options**: `ST_AdoptTrackOrientationOptions`

Initiates a movement for the c component.

**Syntax**

Definition:

```plaintext
METHOD AdoptTrackOrientation
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    constraint       : IPlcDynamicConstraint;
    options          : ST_AdoptTrackOrientationOptions;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td><code>MC_PlanarFeedback</code></td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td><code>IPlcDynamicConstraint</code></td>
<td>Dynamic constraints for this movement.</td>
</tr>
</tbody>
</table>
PLC Libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>ST_AdoptTrackOrientation Options [94]</td>
<td>Options for the rotation.</td>
</tr>
</tbody>
</table>

### 6.2.2.8.12 Halt

**Halt**

- commandFeedback : MC_PlanarFeedback
- constraint : IPlcDynamicConstraint

Initiates a halt.

**Syntax**

**Definition:**

METHOD Halt

VAR_INPUT

commandFeedback : MC_PlanarFeedback;
constraint : IPlcDynamicConstraint;

END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td>IPlcDynamicConstraint</td>
<td>Dynamic constraints for this movement.</td>
</tr>
</tbody>
</table>

### 6.2.2.8.13 Enable

**Enable**

- commandFeedback : MC_PlanarFeedback

Starts enabling the Planar Mover.

**Syntax**

**Definition:**

METHOD Enable

VAR_INPUT

commandFeedback : MC_PlanarFeedback;

END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>
6.2.2.8.14 Disable

Disable

commandFeedback: MC_PlanarFeedback

Starts disabling the Planar Mover.

Syntax

Definition:

METHOD Disable
VAR_INPUT
  commandFeedback: MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.8.15 Reset

Reset

commandFeedback: MC_PlanarFeedback

Starts resetting the Planar Mover.

Syntax

Definition:

METHOD Reset
VAR_INPUT
  commandFeedback: MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.8.16 Update

Update

Updates internal state of the object, must be triggered each cycle.

Syntax

Definition:

METHOD Update
6.2.2.17  SetPosition

**SetPosition**

```
VAR_IN_OUT
commandFeedback : MC_PlanarFeedback;
position : PositionXYC;
END_VAR
```

Sets the position of the Planar Mover. Only possible if the Planar Mover is disabled.

### Syntax

**Definition:**

```
METHOD SetPosition
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    position : PositionXYC;
END_VAR
```

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

#### In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>PositionXYC</td>
<td>New position of the Planar Mover.</td>
</tr>
</tbody>
</table>

6.2.2.18  StartExternalSetpointGeneration

**StartExternalSetpointGeneration**

```
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    options : ST_ExternalSetpointGenerationOptions;
END_VAR
```

Starts the external setpoint generation, the user must supply setpoints from this PLC cycle on every PLC cycle.

### Syntax

**Definition:**

```
METHOD StartExternalSetpointGeneration
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    options : ST_ExternalSetpointGenerationOptions;
END_VAR
```

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>ST_ExternalSetpointGenerationOptions [94]</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>
6.2.2.8.19  StopExternalSetpointGeneration

StopExternalSetpointGeneration

commandFeedback MC_PlanarFeedback

Ends the external setpoint generation, called after last SetExternalSetpoint (in the same PLC cycle).

Syntax

Definition:

METHOD StopExternalSetpointGeneration
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeed-</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>back</td>
<td>[102]</td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.8.20  SetExternalSetpoint

SetExternalSetpoint

setPosition MoverVector
setVelocity MoverVector
setAcceleration MoverVector

Sets the external setpoint for the Planar Mover, 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setPosition</td>
<td>MoverVector</td>
<td>Position that is send to the Planar Mover.</td>
</tr>
<tr>
<td>setVelocity</td>
<td>MoverVector</td>
<td>Velocity that is send to the Planar Mover.</td>
</tr>
<tr>
<td>setAccelera-</td>
<td>MoverVector</td>
<td>Acceleration that is send to the Planar Mover.</td>
</tr>
<tr>
<td>tion</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.8.21  AddToGroup

AddToGroup

commandFeedback MC_PlanarFeedback

Adds the Planar Mover to the given Planar Group.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeed-</td>
<td>MC_PlanarFeedback</td>
<td></td>
</tr>
<tr>
<td>back</td>
<td>MC_PlanarGroup</td>
<td></td>
</tr>
</tbody>
</table>
Syntax

Definition:

```plaintext
METHOD AddToGroup
    VAR_INPUT
        commandFeedback : MC_PlanarFeedback;
    END_VAR
    VAR_IN_OUT
        group            : MC_PlanarGroup;
    END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

**In/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>MC_PlanarGroup</td>
<td>The Planar Group that the Planar Mover joins.</td>
</tr>
</tbody>
</table>

### 6.2.2.8.22 RemoveFromGroup

```plaintext
REM oveFromGroup
commandFeedback MC_PlanarFeedback
```

Removes the Planar Mover from its current Planar Group, i.e. disables collision checks.

Syntax

Definition:

```plaintext
METHOD RemoveFromGroup
    VAR_INPUT
        commandFeedback : MC_PlanarFeedback;
    END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

### 6.2.2.8.23 GetPlanarObjectInfo

```plaintext
Ge tPlanarObjectInfo
PlanarObjectInfo GetPlanarObjectInfo
```

Returns mover object info (type: mover, id: OID of nc mover).

Syntax

Definition:

```plaintext
METHOD GetPlanarObjectInfo : PlanarObjectInfo
```

---

TF5430 Version: 1.3
Return value

PlanarObjectInfo [93]

6.2.2.9 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear [123]</td>
<td>Clears the geometric information of the Planar Track.</td>
</tr>
<tr>
<td>AppendPosition [123]</td>
<td>Appends a position to the Planar Track.</td>
</tr>
<tr>
<td>AppendLine [124]</td>
<td>Appends a line to the Planar Track.</td>
</tr>
<tr>
<td>AppendCircle [124]</td>
<td>Appends a circular arc to the Planar Track.</td>
</tr>
<tr>
<td>CloseLoop [125]</td>
<td>Closes the loop of the Planar Track, no further part can be appended.</td>
</tr>
<tr>
<td>StartFromTrack [125]</td>
<td>Sets the other Planar Tracks endpoint as start point of this Planar Track, transition is smooth. The other Planar Track is blocked for further changes (until it is cleared).</td>
</tr>
<tr>
<td>EndAtTrack [126]</td>
<td>Appends a smooth transition from the end of this Planar Track to the other Planar Tracks start point. The Planar Track is blocked for further changes (until it is cleared).</td>
</tr>
<tr>
<td>Enable [126]</td>
<td>Starts enabling the Planar Track.</td>
</tr>
<tr>
<td>Disable [127]</td>
<td>Starts disabling the Planar Track.</td>
</tr>
<tr>
<td>Reset [127]</td>
<td>Starts resetting the Planar Track.</td>
</tr>
<tr>
<td>GetArcLengthClosestTo [128]</td>
<td>Calculate the arc length value where the Planar Track is closest to a geometry's center point.</td>
</tr>
<tr>
<td>GetPositionAt [128]</td>
<td>Get a position on the Planar Track at a specific arc length value.</td>
</tr>
<tr>
<td>GetLength [129]</td>
<td>Returns the Planar Tracks length, -1 return value indicates no connection to Nc Track.</td>
</tr>
<tr>
<td>Update [129]</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
<tr>
<td>AddToGroup [129]</td>
<td>Adds the Planar Track to the given Planar Group.</td>
</tr>
<tr>
<td>RemoveFromGroup [130]</td>
<td>Removes the Planar Track from its current Planar Group, i.e. disables collision checks.</td>
</tr>
</tbody>
</table>

Required License

TC3 Planar Motion Base

System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.12</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.2.2.9.1 Clear

Clear

commandFeedback MC_PlanarFeedback

Clears the geometric information of the Planar Track.

Syntax

Definition:

METHOD Clear
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.9.2 AppendPosition

AppendPosition

commandFeedback MC_PlanarFeedback

position PositionXYC

Appends a position to the Planar Track.

Syntax

Definition:

METHOD AppendPosition
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
    position : PositionXYC;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>PositionXYC</td>
<td>Position that is the new endpoint of the Planar Track.</td>
</tr>
</tbody>
</table>
6.2.2.9.3 AppendLine

<table>
<thead>
<tr>
<th>commandFeedback</th>
<th>MC_PlanarFeedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>PositionXYC</td>
</tr>
<tr>
<td>end</td>
<td>PositionXYC</td>
</tr>
</tbody>
</table>

Appends a line to the Planar Track.

**Syntax**

**Definition:**

METHOD AppendLine

VAR_INPUT

    commandFeedback : MC_PlanarFeedback;

END_VAR

VAR_IN_OUT

    start : PositionXYC;
    end  : PositionXYC;

END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

**In/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>PositionXYC</td>
<td>Start position of the line.</td>
</tr>
<tr>
<td>end</td>
<td>PositionXYC</td>
<td>End position of the line, this position is the new endpoint of the Planar Track.</td>
</tr>
</tbody>
</table>

6.2.2.9.4 AppendCircle

<table>
<thead>
<tr>
<th>commandFeedback</th>
<th>MC_PlanarFeedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>clockwise</td>
<td>BOOL</td>
</tr>
<tr>
<td>start</td>
<td>PositionXYC</td>
</tr>
<tr>
<td>end</td>
<td>PositionXYC</td>
</tr>
<tr>
<td>center</td>
<td>PositionXY</td>
</tr>
</tbody>
</table>

Appends a circular arc to the Planar Track.

**Syntax**

**Definition:**

METHOD AppendCircle

VAR_INPUT

    commandFeedback : MC_PlanarFeedback;
    clockwise       : BOOL;

END_VAR

VAR_IN_OUT

    start : PositionXYC;
    end  : PositionXYC;
    center : PositionXY;

END_VAR
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>clockwise</td>
<td>BOOL</td>
<td>Indicates if the clockwise circle is appended.</td>
</tr>
</tbody>
</table>

## In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>PositionXYC</td>
<td>Start position of the circular arc.</td>
</tr>
<tr>
<td>end</td>
<td>PositionXYC</td>
<td>End position of the circular arc, this position is the new endpoint of the Planar Track.</td>
</tr>
<tr>
<td>center</td>
<td>PositionXY</td>
<td>Center of the circular arc.</td>
</tr>
</tbody>
</table>

6.2.2.9.5 CloseLoop

**CloseLoop**

```
   commandFeedback  MC_PlanarFeedback
```

Closes the loop of the Planar Track, no further part can be appended.

**Syntax**

Definition:

```
METHOD CloseLoop
VAR_INPUT
   commandFeedback : MC_PlanarFeedback;
END_VAR
```

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.9.6 StartFromTrack

**StartFromTrack**

```
   commandFeedback  MC_PlanarFeedback
   track           MC_PlanarTrack
```

Sets the other Planar Tracks 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 : MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
   track                   : MC_PlanarTrack;
END_VAR
```
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

## In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>track</td>
<td>MC_PlanarTrack [122]</td>
<td>The other Planar Track.</td>
</tr>
</tbody>
</table>

### 6.2.2.9.7 EndAtTrack

```plaintext
ENDAtTrack
  commandFeedback MC_PlanarFeedback
  track MC_PlanarTrack
```

Appends a smooth transition from the end of this Planar Track to the other Planar Tracks start point. The Planar Track is blocked for further changes (until it is cleared).

#### Syntax

**Definition:**

```plaintext
METHOD EndAtTrack
  VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
  END_VAR
  VAR_IN_OUT
    track : MC_PlanarTrack;
  END_VAR
```

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

## In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>track</td>
<td>MC_PlanarTrack [122]</td>
<td>The other Planar Track.</td>
</tr>
</tbody>
</table>

### 6.2.2.9.8 Enable

```plaintext
Enable
  commandFeedback MC_PlanarFeedback
```

Starts enabling the Planar Track.

#### Syntax

**Definition:**
METHOD Enable
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.9.9 Disable

Disable

commandFeedback : MC_PlanarFeedback

Starts disabling the Planar Track.

Syntax

Definition:
METHOD Disable
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.2.2.9.10 Reset

Reset

commandFeedback : MC_PlanarFeedback

Starts resetting the Planar Track.

Syntax

Definition:
METHOD Reset
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>
6.2.2.9.11 GetArcLengthClosestTo

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometry</td>
<td>IPlcGeometry2D</td>
<td>The geometry to check the arc length for.</td>
</tr>
</tbody>
</table>

Return value

LREAL

6.2.2.9.12 GetPositionAt

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arcLength</td>
<td>LREAL</td>
<td>Arc length value where the position is evaluated.</td>
</tr>
</tbody>
</table>

In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>PositionXYC</td>
<td>The position at the specified arc parameter.</td>
</tr>
</tbody>
</table>
6.2.2.9.13  GetLength

Returns the Planar Tracks length, -1 return value indicates no connection to Nc Track.

Syntax
Definition:
METHOD GetLength : LREAL

Return value
LREAL

6.2.2.9.14  GetPlanarObjectInfo

Returns track object info (type: track, id: OID of nc track).

Syntax
Definition:
METHOD GetPlanarObjectInfo : PlanarObjectInfo

Return value
PlanarObjectInfo [93]

6.2.2.9.15  Update

Updates internal state of the object, must be triggered each cycle.

Syntax
Definition:
METHOD Update

6.2.2.9.16  AddToGroup

Adds the Planar Track to the given Planar Group.
## Syntax

**Definition:**

```plaintext
METHOD AddToGroup
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
  group : MC_PlanarGroup;
END_VAR
```

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

### In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>MC_PlanarGroup</td>
<td>The Planar Group that the mover joins.</td>
</tr>
</tbody>
</table>

### 6.2.2.9.17 RemoveFromGroup

**Syntax**

**Definition:**

```plaintext
METHOD RemoveFromGroup
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

**6.2.2.10 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>Clears the TrackTrail.</td>
</tr>
<tr>
<td>AddTrack</td>
<td>Adds a track to the TrackTrail. The track should start at the end vertex of the currently last track.</td>
</tr>
</tbody>
</table>
Required License
TC3 Planar Motion Base

System Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.1.4024.17</td>
<td>PC or CX (x64)</td>
<td>Tc3_Mc3PlanarMotion, Tc3_Physics</td>
</tr>
<tr>
<td>Advanced Motion Pack V3.1.10.44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.2.10.1 Clear

Clear

Clears the TrackTrail.

Syntax

Definition:

METHOD Clear

6.2.2.10.2 AddTrack

AddTrack

Add 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>track</td>
<td>MC_PlanarTrack [122]</td>
<td>The track to be added to the end of the TrackTrail.</td>
</tr>
</tbody>
</table>