# Table of contents

1 Foreword ........................................................................................................................................ 5  
   1.1 Notes on the documentation ....................................................................................................... 5  
   1.2 Safety instructions .......................................................................................................................... 6  
2 Introduction ......................................................................................................................................... 7  
3 TF5430 TC3 Planar Motion - overview of the new features .............................................................. 8  
4 States and modes ................................................................................................................................... 9  
   4.1 Planar objects state diagram ........................................................................................................... 9  
   4.2 Planar mover command diagram .................................................................................................... 10  
   4.3 Planar track operation modes ......................................................................................................... 11  
5 Planar Motion components .................................................................................................................. 13  
   5.1 Planar mover .................................................................................................................................. 13  
      5.1.1 Configuration .............................................................................................................................. 13  
      5.1.2 Example: "Creating and moving Planar movers" ...................................................................... 16  
      5.1.3 Example: "Creating and moving a Planar mover with auxiliary axes" ................................ 19  
      5.1.4 Example "Creating and moving a Planar mover with External Setpoint Generation" .... 22  
      5.1.5 Limits and options of the motion commands ............................................................................ 26  
   5.2 Planar track .................................................................................................................................... 26  
      5.2.1 Configuration .............................................................................................................................. 26  
      5.2.2 Track networks and collision avoidance .................................................................................. 29  
      5.2.3 Example "Joining and moving a Planar mover on the track" ............................................... 31  
      5.2.4 Example "Connecting Planar tracks to a network" .................................................................. 36  
   5.3 Planar group ................................................................................................................................... 41  
      5.3.1 Configuration .............................................................................................................................. 41  
      5.3.2 Example: "Creating and moving Planar movers with group" ................................................. 43  
   5.4 Planar environment ......................................................................................................................... 48  
      5.4.1 Configuration .............................................................................................................................. 48  
      5.4.2 Example "Configuring the stator area and boundary" ............................................................. 51  
   5.5 Example: "Creating and moving Planar movers with track and group" ...................................... 54  
   5.6 Planar Feedback ............................................................................................................................. 58  
      5.6.1 Example "Creating a Planar mover and Planar Feedback" .................................................... 59  
      5.6.2 Example "Planar motion components: averting collision" ...................................................... 62  
6 PLC Libraries ...................................................................................................................................... 67  
   6.1 Inserting libraries ............................................................................................................................. 67  
   6.2 Tc3_Physics API ............................................................................................................................... 67  
      6.2.1 Function Blocks ........................................................................................................................... 67  
   6.3 Tc3_Mc3PlanarMotion API ............................................................................................................. 81  
      6.3.1 Data Types ................................................................................................................................... 81  
      6.3.2 Function Blocks ........................................................................................................................... 86
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®, 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:


with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

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

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

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DANGER</strong></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><strong>WARNING</strong></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><strong>CAUTION</strong></td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><strong>NOTE</strong></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.
2 Introduction

The TC3 Planar Motion software package TF5430 is installed together with the software package TF5400.

Target system
Windows 7/8/10 (64Bit only)

TC3 Planar Motion Base
The TF5430 TC3 Planar Motion software combines a wide range of functionalities for controlling XPlanar movers and enables efficient and intelligent implementation of individual XPlanar applications. TF5430 TC3 Planar Motion is part of TF5890 TC3 XPlanar. All associated function blocks are included in the library Tc3_Mc3PlanarMotion, which is to be used in combination with the library Tc3_Physics.
3 TF5430 TC3 Planar Motion - overview of the new features

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

- **Error**
  - Can be triggered in every state (except Resetting).
States and modes

**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 five different command modes that indicate what type of command the mover executes: OnTrack, LeavingTrack, JoiningTrack, FreeMovement and ExternalSetpointGeneration.

![Planar mover command diagram](image)

Except for ExternalSetpointGeneration mode, when the mover is in a group, the mover reserves its geometry in all modes so that collisions are avoided. In addition, for external setpoint generation, there is a relative mode in addition to the normal absolute mode. This can be activated by the user while the mover is in the OnTrack, JoiningTrack, LeavingTrack or FreeMovement modes. Even in the relative mode of external setpoint generation, collision avoidance may not be fully functional.

**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.

**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 the mover has been enabled, it is automatically in this command 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.

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.

4.3 Planar track operation modes

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

Moving

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

Standing

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

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

**Uninitialized**

The track is not usable by movers in the Uninitialized mode. It does not have a finished geometric description yet. When the user creates and enables this geometric description, the track switches to Standing mode.
5  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.

   ![Insert Motion Configuration dialog box]

   ✓ You have created an MC project.

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

<table>
<thead>
<tr>
<th>Search:</th>
<th>Name:</th>
<th>Move1 [Planar Mover]</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td></td>
<td>Beckhoff Automation GmbH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Motion Control</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Planar Mover [Module]</td>
<td></td>
</tr>
</tbody>
</table>

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

- **Parameters (Init)** should be put into simulation mode before parameterizing if no hardware driver is linked.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>CS</th>
<th>Unit</th>
<th>Type</th>
<th>PID</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation mode</td>
<td>FALSE</td>
<td></td>
<td></td>
<td>BOOL</td>
<td>0x006909B1</td>
<td>Specifies if mover is in simulation mode or not.</td>
</tr>
<tr>
<td>Mover width</td>
<td>1550</td>
<td></td>
<td>mm</td>
<td>REAL</td>
<td>0x006909B2</td>
<td>Width that is used for internal collision checks.</td>
</tr>
<tr>
<td>Mover height</td>
<td>1550</td>
<td></td>
<td>mm</td>
<td>REAL</td>
<td>0x006909B3</td>
<td>Height that is used for internal collision checks.</td>
</tr>
<tr>
<td>Initial position</td>
<td>...</td>
<td></td>
<td>mm</td>
<td></td>
<td>0x006909B4</td>
<td>Mover position, can be set for disabled mover.</td>
</tr>
<tr>
<td>Maximum Dynamics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum dynamic XY</td>
<td>...</td>
<td></td>
<td>mm per s, s...</td>
<td>0x006909B5</td>
<td>Maximum dynamic of the mover in the XY plane.</td>
<td></td>
</tr>
</tbody>
</table>
| Maximum dynamic C     | ...   |    | * per s, s...
| Maximum dynamic Z     | ...   |    | mm per s, s...
| Maximum dynamic A     | ...   |    | * per s, s...
| Maximum dynamic B     | ...   |    | * per s, s...
| Default Dynamics      |       |    |       |       |     |                                              |
| Default dynamic XY    | ...   |    | mm per s, s...
| Default dynamic C     | ...   |    | * per s, s...
| Default dynamic Z     | ...   |    | mm per s, s...
| Default dynamic A     | ...   |    | * per s, s...
| Default dynamic B     | ...   |    | * per s, s...
| Monitoring            |       |    |       |       |     |                                              |
| Position Lag Monitoring Enabled | TRUE |    |       | BOOL  | 0x0068011A | A vector of six numeric values corresponding to...
| Maximum Position Lag Value | ... |    | mm    |       | 0x0068008E | A vector of six numeric values corresponding to...
| Maximum Position Lag Filter Time | ... |    | s     |       | 0x0068008F | A vector of six numeric values corresponding to...

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. 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.

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

<table>
<thead>
<tr>
<th>Object</th>
<th>Parameter (int)</th>
<th>Parameter (Online)</th>
<th>Data Area</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (0)</td>
<td>ioToMc</td>
<td>InputDst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>PlcToMc</td>
<td>InputDst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (0)</td>
<td>McToIoC</td>
<td>OutputSrc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 (0)</td>
<td>McToPlc</td>
<td>OutputSrc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 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 [13].
- 1. Create a Planar mover.
- 2. Put "Parameters (Init)" into simulation mode.

Creating a PLC

- In order to control the mover, a PLC must be created from which the user can issue commands to the mover.
For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see \[ Inserting libraries \]

3. Create a mover via **MAIN** ("MC_PlanarMover").

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

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

```plaintext
CASE state OF
  0:
    mover.Enable(0);
```
Planar Motion components

```c
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
```

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

**Sending the command**

4. To send the command, you must call the mover cyclically with its update method after END_CASE:
   ```c
   mover.Update();
   ```

The PLC should be built once, so that the system knows the PLC mover and it can be linked to the Planar mover in the MC project.

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

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

**Activating and starting the project**

1. Activate the configuration via the button in the menu bar .
2. Log in the PLC via the button in the menu bar .
3. Start the PLC via the Play button in the menu bar.

\( \Rightarrow \) At the end of the state machine (state=3), the mover is in the desired position.
5.1.3 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 [13].
  1. Create a Planar mover.
  2. Put "Parameters (Init)" into simulation mode.

Creating a PLC

- In order to control the mover, a PLC must be created from which the user can issue commands to the mover.
For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [67].

3. Create a mover via MAIN ("MC_PlanarMover").

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

In this simple example you have created a state variable for a state machine and some target positions for a travel command of the mover, with which a sequence can subsequently be programmed in the MAIN:
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);
      mover.MoveC(0, target_c, 0);
      mover.MoveZ(0, target_z, 0);
      state := 3;
END_CASE

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

Sending the command

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

The PLC should be built once, so that the system knows the PLC mover and it can be linked to the Planar mover in the MC project.

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

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

Activating and starting the project

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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 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 [13].

1. Create a Planar mover.
2. Put "Parameters (Init)" into simulation mode.

Creating a PLC

- In order to control the mover, a PLC must be created from which the user can issue commands to the mover.
For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [P.67].

3. Create a mover via **MAIN** ("MC_PlanarMover").

![Image of TwinCAT Project Explorer]

This represents the mover in the PLC.

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

In this simple example you have created a state variable for a state machine and variables for the external setpoint, with which a sequence can subsequently be programmed in the 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).

**Sending the command**

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

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

The PLC should be built once, so that the system knows the PLC mover and it can be linked to the Planar mover in the MC project.

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

![PLC interface]

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

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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.
5.1.5 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. External setpoint generation is not covered in this section. The following applies to the rest of the motion commands. The first parameter of the method call is always the feedback for the command that the user transfers. If he transfers a "0", this implies that he does not want to have (or use) feedback. The next one or two parameters describe the destination of the motion and they cannot be completely omitted. The next parameters are the dynamic limits that should be observed during motion. If the user transfers a "0" here, the default values are used (TCOM parameters of the mover in the MC Project). The last parameter is the option object, which differs depending on the command.

Limits

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

Options

The options vary depending on the command:

MoveToPosition/JoinTrack/LeaveTrack: The only option with 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.

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. In the **MC Project** created, select **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.

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

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.

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

1. Create two Planar movers.
2. Put "Parameters (Init)" into simulation mode.
3. Change the start position of the second mover to \( x = 240 \).

**Create the Planar track**

4. Add the Planar track via **Groups > Add New Item...**, see **Configuration [26]**.

**Creating a PLC**

✓ In order to control the movers and the track, a PLC must be created from which the user can issue commands to the mover and the track.

For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see **Inserting libraries [67]**.

5. Create two movers ("MC_PlanarMover") and an "MC_PlanarTrack" via **MAIN**.
These variables represent the movers and track in the PLC.

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

In this simple example you have created a state variable for a state machine and two auxiliary positions for the track, with which a sequence can subsequently be programmed in the MAIN.

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

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

Sending the command

6. 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();

The PLC should be built once, so that the system knows the PLC movers and the PLC track and they can then be linked to the Planar mover in the MC project.

7. 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

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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 "Connecting Planar tracks to a network"

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

Create the Planar track

1. Add four Planar tracks via Groups > Add New Item..., see Configuration [26].

Creating a PLC

✓ A PLC must be created in order to design the network.
For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [p. 67].

2. Create four "MC_PlanarTracks" via MAIN.

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

CASE state OF
  0:
    pos1.SetValuesXY(50, 0);
    pos2.SetValuesXY(450, 0);
    track_one.AppendLine(0, pos1, pos2);
    pos1.SetValuesXY(500, 50);
    pos2.SetValuesXY(500, 450);
    track_one.AppendLine(0, pos1, pos2);
    pos1.SetValuesXY(450, 500);
    pos2.SetValuesXY(50, 500);
    track_one.AppendLine(0, pos1, pos2);
    state := 1;
  1:
    pos1.SetValuesXY(0, 450);
    pos2.SetValuesXY(0, 50);
    track_two.StartFromTrack(0,track_one);
    track_two.AppendLine(0, pos1, pos2);
    track_two.EndAtTrack(0,track_one);
    state := 2;
  2:
    pos1.SetValuesXY(0, 500);
    pos2.SetValuesXY(-200, 500);
    track_three.StartFromTrack(0,track_one);
    track_three.AppendLine(0, pos1, pos2);
    state := 3;
  3:
    pos1.SetValuesXY(0, 550);
    pos2.SetValuesXY(0, 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
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.

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**

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

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

The PLC should be built once, so that the system knows the PLC tracks and they can then be linked with the objects in the MC project.

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

.expression

The tracks must each be linked separately via the following dialog boxes.
Activating and starting the project

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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.
5.3 Planar group

The Planar group is a software object that prevents collisions between Planar movers as well as collisions of Planar movers with the edge of the stator surface on the two-dimensional XPlanar stator surface. To do this, the 2D surfaces 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 areas 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. In the MC Project created, select 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)**

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

- **OK**
- **Cancel**
- **Multiplier:** 1
- **Insert instance**
- **Reload**

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

**Parameters (Init)** should be put into simulation mode before parameterizing if no hardware driver is linked.

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

Parameters (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 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 [13].
  1. Create two Planar movers.
  2. Put "Parameters (Init)" into simulation mode.
  3. Change the start position of the second mover to \( x = 240 \).

Creating a Planar group
  4. Add the Planar group via Groups > Add New Item..., see Configuration [41].

Creating a PLC
  ✓ A PLC must be created in order to control the movers and the group.
For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [67].

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

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

In this simple example you have created a state variable for a state machine and two auxiliary positions for the "MoveToPosition" commands of the movers, with which a sequence can subsequently be programmed in the MAIN.
CASE state OF
 0:
    mover_one.Enable(0);
    mover_two.Enable(0);
    state := 1;
 1:
    IF mover_one.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled AND mover_two.MCTOPLC.STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 2;
    END_IF
 2:
    group.Enable(0);
    state := 3;
 3:
    IF group.MCTOPLC_STD.State = MC_PLANAR_STATE.Enabled THEN
      state := 4;
    END_IF
 4:
    mover_one.AddToGroup(0, group);
    mover_two.AddToGroup(0, group);
    state := 5;
 5:
    IF mover_one.MCTOPLC_STD.GroupOID = group.MCTOPLC_STD.GroupOID AND mover_two.MCTOPLC_STD.GroupOID = group.MCTOPLC_STD.GroupOID THEN
      state := 6;
    END_IF
 6:
    pos1.SetValuesXY(0, 240);
    pos2.SetValuesXY(0, 0);
    mover_one.MoveToPosition(0, pos1, 0, 0);
    mover_two.MoveToPosition(0, pos2, 0, 0);
    state := 7;
END_CASE

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

Sending the command

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

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

The PLC should be built once, so that the system knows the PLC movers and the PLC group and they can then be linked with the objects in the MC project.

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

```
PLC
Untitled1 Project
  ▼ External types
  ▼ References
    ▼ DUs
    ▼ GVs
    ▼ POUs
  ▼ MAIN (PRG)
```

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

1. Double-click Mover 1 first, then Mover 2.
The group must be linked separately via the following dialog boxes.
Activating and starting the project

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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 [p. 13].
Planar Motion components

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.
3. In the **MC Project** created, select **Groups > Add New Item**...

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

   ![Image of the MC Project creation process]

   ![Image of the Planar Environment selection]

   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.

<table>
<thead>
<tr>
<th>Object Id:</th>
<th>0x05120010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Name:</td>
<td>Group1 (Planar Environment)</td>
</tr>
<tr>
<td>Typename:</td>
<td>Planar Environment</td>
</tr>
<tr>
<td>GUID:</td>
<td>9A40A34B-DD2F-4271-8FEE-3ADE2271DB79</td>
</tr>
<tr>
<td>Class Id:</td>
<td>050300CB-0000-0000-F000-000000000064</td>
</tr>
<tr>
<td>Class Factory:</td>
<td>TcNc3</td>
</tr>
<tr>
<td>TMI/TMC:</td>
<td>C:\TwinCAT\3.1\Config\Modules\TcNc3.mtc</td>
</tr>
<tr>
<td>Parent Id:</td>
<td>0x05100010</td>
</tr>
<tr>
<td>Init Sequence:</td>
<td>PSO</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Object</th>
<th>Parameter (Init)</th>
<th>Parameter (Online)</th>
<th>Data Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area No</td>
<td>Name</td>
<td>Type</td>
<td>Size</td>
</tr>
<tr>
<td>1 (0)</td>
<td>McTcToPlc</td>
<td>OutputSrc</td>
<td>16</td>
</tr>
</tbody>
</table>

### 5.4.2 Example "Configuring the stator area and boundary"

Using this guide you will create a TwinCAT project that contains a Planar environment and you will configure its stator area and boundary.

**Creating a Planar environment**

1. Create a Planar environment, see Configuration [48].

**Creating a PLC**

- In order to create the geometry of the environment, a PLC must be created from which the user can send commands to the environment.

For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [67].

2. Create an "MC_PlanarEnvironment" via **MAIN**.
PLANAR MOTION COMPONENTS

This represents the environment in the PLC.

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

In this example you have created a state variable for a state machine, with which a sequence can subsequently be programmed in the MAIN.

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

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.

Sending the command

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

environment.Update();
The PLC should be built once, so that the system knows the PLC environment and it can be linked to the Planar environment in the MC project.

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

The Planar environment can then be linked in the "MC Project".
Activating and starting the project

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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 [13].
1. Create two Planar movers.
2. Put "Parameters (Init)" into simulation mode.
3. Change the start position of the second mover to x = 240.

Create a Planar track and a Planar group

4. Add the Planar track via Groups > Add New Item..., see Configuration [26].
5. Proceed in the same way for the Planar group.

Creating a PLC

✓ In order to control the movers, the track and the group, a PLC must be created from which the user can issue commands to the mover.

For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [67].

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

PROGRAM MAIN
VAR
    mover_one, mover_two : MC_PlanarMover;
    track : MC_PlanarTrack;
    group : MC_PlanarGroup;
    state : UDINT;

These variables represent the movers, the track and the group in the PLC.
In this simple example you have created a state variable for a state machine and two auxiliary positions for the track. In addition, a feedback is created to which any command can be given and which provides detailed information about the command execution over the execution time. With this, a sequence can now be programmed in the MAIN:

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

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).

Sending the command

7. To send the command, you must call the mover, the track and the group cyclically with their update method after the END_CASE:
mover_one.Update();
mover_two.Update();
track.Update();
group.Update();
feedback.Update();

The PLC should be built once, so that the system knows the PLC mover and it can be linked to the Planar mover in the MC project.

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

![PLC build](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**

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

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

3. 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.
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 to use Planar Feedback, 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 [13].
1. Create a Planar mover.
2. Put "Parameters (Init)" into simulation mode.

Creating a PLC

✓ In order to control the mover and use a Planar feedback, a PLC must be created from which the user can issue commands to the mover.

For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [67].

3. Create a mover ("MC_PlanarMover") and a Planar Feedback ("MC_PlanarFeedback") via MAIN.
These represent the mover and the Planar Feedback in the PLC.

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

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

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

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

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

Sending the command

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:

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

The PLC should be built once, so that the system knows the PLC mover and it can be linked to the Planar mover in the MC project.

5. To build, use the path PLC > Untitled1 > Untitled1 Project > Build.
Subsequently, the Planar mover in the "MC Project" (double-click) can be linked with the Link To PLC... button on the Settings tab.

Activating and starting the project

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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.
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 [13].

1. Create a Planar mover.
2. Put “Parameters (Init)” into simulation mode.

Creating a Planar environment

3. Create a Planar environment, see Configuration [48].

Creating a Planar group

4. Create a Planar group, see Configuration [41].

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.
For the PLC project, you need to add the "Tc3_Physics" and "Tc3_Mc3PlanarMotion" libraries, see Inserting libraries [p. 67].

5. Create an "MC_PlanarMover" and an "MC_PlanarEnvironment" via MAIN.

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:
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

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

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

The PLC should be built once, so that the system knows the PLC mover and it can be linked to the Planar mover in the MC project.

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

8. 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**

1. Activate the configuration via the button in the menu bar.
2. Log in the PLC via the button in the menu bar.
3. 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.
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 two 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.

6.2 Tc3_Physics API

6.2.1 Function Blocks

6.2.1.1 Dynamics

6.2.1.1.1 DynamicConstraint_Path

One dimensional dynamic constraint along a path, ignoring non tangential effects.
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>SetValuesVADJ [68]</td>
<td>Set the dynamic limits of this instance.</td>
</tr>
</tbody>
</table>

Required License

TC3 Physics 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>

SetValuesVADJ

Set the dynamic limits of this instance.

Syntax

Definition:

METHOD SetValuesVADJ
VAR_INPUT
   v : LREAL;
   a : LREAL;
   d : LREAL;
   j : LREAL;
END_VAR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>LREAL</td>
<td>maximal velocity.</td>
</tr>
<tr>
<td>a</td>
<td>LREAL</td>
<td>maximal acceleration.</td>
</tr>
<tr>
<td>d</td>
<td>LREAL</td>
<td>maximal deceleration.</td>
</tr>
<tr>
<td>j</td>
<td>LREAL</td>
<td>maximal jerk.</td>
</tr>
</tbody>
</table>

6.2.1.2 Spatial

6.2.1.2.1 Positions

PositionXY

Position in the 2D Cartesian space.

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>SetZero</td>
<td>Set the coordinates of this position to '0.0'.</td>
</tr>
<tr>
<td>SetValues</td>
<td>Set the coordinates of this position.</td>
</tr>
<tr>
<td>SetValuesXY</td>
<td>Set the xy-coordinates of this position.</td>
</tr>
<tr>
<td>SetValuesRP</td>
<td>Set the components of this position in polar coordinates.</td>
</tr>
</tbody>
</table>
### PLC Libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRadius [71]</td>
<td>Get the distance from the origin.</td>
</tr>
<tr>
<td>GetPhi [72]</td>
<td>Get the polar angle scaled in degrees [°].</td>
</tr>
<tr>
<td>Invert [72]</td>
<td>Invert this position.</td>
</tr>
<tr>
<td>ConcatenateWith [73]</td>
<td>Concatenate with a 2nd position.</td>
</tr>
<tr>
<td>ShiftByXY [74]</td>
<td>Displace the components of this position.</td>
</tr>
</tbody>
</table>

#### Required License

TC3 Physics 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>

### SetZero

**SetZero**

Set the coordinates of this position to '0.0'.

**Syntax**

**Definition:**

METHOD SetZero

**Example: PositionXY.SetZero**

**PLC declaration**

```plaintext
VAR
  position : PositionXY;
VAR_END
```

**PLC implementation**

```plaintext
position.x := 0.3;
posiisión.y := 0.5;
position.SetZero();
```

**Expected result**

```plaintext
position.x = 0
position.y = 0
```

### SetValues

**SetValues**

Set the coordinates of this position.

**Syntax**

**Definition:**

METHOD SetValues

**Example:**

```plaintext
VAR
  positionXY : PositionXY;
VAR_END

positionXY.SetValues(0.3, 0.5);
```
METHOD SetValues
VAR_IN_OUT
    positionXY : PositionXY;
END_VAR

**In/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>positionXY</td>
<td>PositionXY</td>
<td>The coordinate values to set.</td>
</tr>
</tbody>
</table>

**Example: PositionXY.SetValues**

**PLC declaration**

```
VAR
    position0 : PositionXY;
    position1 : PositionXY;
VAR_END
```

**PLC implementation**

```plaintext
position0.x := 5.6;
position0.y := 0.2;
position1.SetValues (position0);
```

**Expected result**

position1.x = 5.6
position1.y = 0.2

**SetValuesXY**

```
SetValuesXY
x LREAL
y LREAL
```

Set the xy-coordinates of this position.

**Syntax**

**Definition:**

```
METHOD SetValuesXY
VAR_INPUT
    x : LREAL;
    y : LREAL;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>LREAL</td>
<td>x-component of the position.</td>
</tr>
<tr>
<td>y</td>
<td>LREAL</td>
<td>y-component of the position.</td>
</tr>
</tbody>
</table>

**Example: PositionXY.SetValuesXY**

**PLC declaration**

```
VAR
    position : PositionXY;
    x : LREAL := 4.3;
    y : LREAL := 2.5;
VAR_END
```

**PLC implementation**
position.x := 0;
position.y := 0;
position.SetValuesXY (x, y);

Expected result
position.x = 4.3
position.y = 2.5

**SetValuesRP**

Set the components of this position in polar coordinates.

**Syntax**

**Definition:**

METHOD SetValuesRP
VAR_INPUT
  radius : LREAL;
  phi    : LREAL;
END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>LREAL</td>
<td>Distance from the origin.</td>
</tr>
<tr>
<td>phi</td>
<td>LREAL</td>
<td>Polar angle of this position in degrees [°].</td>
</tr>
</tbody>
</table>

**Example: PositionXY.SetValuesRP**

**PLC declaration**

VAR
  position : PositionXY;
  r : LREAL := 1;
  p : LREAL := 15;
VAR_END

**PLC implementation**

position.x := 0;
position.y := 0;
position.SetValuesRP (r, p);

**Expected result**

position.x = 0.966
position.y = 0.259

**GetRadius**

Get the distance from the origin.

**Syntax**

**Definition:**
METHOD GetRadius : LREAL

Return value

LREAL

Example: PositionXY.GetRadius

PLC declaration

VAR
    position : PositionXY;
    distance : LREAL;
VAR_END

PLC implementation

position.SetValues (5.6, 0.2);
distance := position.GetRadius ();

Expected result

distance = 5.604

GetPhi

Get the polar angle scaled in degrees [°].

Syntax

Definition:

METHOD GetPhi : LREAL

Return value

LREAL

Example: PositionXY.GetPhi

PLC declaration

VAR
    position : PositionXY;
    polarAngle : LREAL;
VAR_END

PLC implementation

position.SetValues (5.6, 0.2);
polarAngle := position.GetPhi ();

Expected result

polarAngle = 2.045

Invert

Invert this position.
Syntax

Definition:

METHOD Invert

Example: PositionXY.Invert

PLC declaration

VAR
  position : PositionXY;
VAR_END

PLC implementation

position.SetValuesXY (1, 3);
position.Invert ();

Expected result

position.x = -1
position.y = -3

ConcatenateWith

ConcatenateWith

Concatenate with a 2nd position.

Syntax

Definition:

METHOD ConcatenateWith

VAR_IN_OUT
  position : PositionXY;
END_VAR

In/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>PositionXY</td>
<td>The 2nd position to concatenate with.</td>
</tr>
</tbody>
</table>

Example: PositionXY.ConcatenateWith

PLC declaration

VAR
  position0 : PositionXY;
  position1 : PositionXY;
VAR_END

PLC implementation

position0.SetValuesXY (0.5, 1.2);
position1.SetValuesXY (0.3, 0.5);
position0.ConcatenateWith (position1);

Expected result

position0.x = 0.8
position0.y = 1.7
**ShiftByXY**

Displace the components of this position.

**Syntax**

**Definition:**

METHOD ShiftByXY
VAR_INPUT
    - dx : LREAL;
    - dy : LREAL;
END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx</td>
<td>LREAL</td>
<td>Shift of the x-component of this position.</td>
</tr>
<tr>
<td>dy</td>
<td>LREAL</td>
<td>Shift of the y-component of this position.</td>
</tr>
</tbody>
</table>

**Example: PositionXY.ShiftByXY**

**PLC declaration**

```
VAR
    position : PositionXY;
    dx : LREAL := 2.0;
    dy : LREAL := 0.0;
VAR_END
```

**PLC implementation**

```
position.SetValuesXY (0, 1);
position.ShiftByXY (dx, dy);
```

**Expected result**

```
position.x = 2
position.y = 1
```

**PositionXYC**

A position in the xy-plane with an in-plane direction c.

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>SetZero</td>
<td>Set the components of this vector to '0.0'.</td>
</tr>
<tr>
<td>SetValues</td>
<td>Set the components of this position.</td>
</tr>
<tr>
<td>SetValuesXY</td>
<td>Set the spatial components of this position.</td>
</tr>
<tr>
<td>SetValuesXYC</td>
<td>Set the components of this position.</td>
</tr>
<tr>
<td>Invert</td>
<td>Invert this position.</td>
</tr>
<tr>
<td>ConcatenateWith</td>
<td>Multiply this position by a 2nd from the right.</td>
</tr>
<tr>
<td>StepByLocalXY</td>
<td>Apply a local step in respect to the local coordinate frame without changing the orientation.</td>
</tr>
</tbody>
</table>
### PLC Libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShiftByXY [79]</td>
<td>Displace the components of this position.</td>
</tr>
<tr>
<td>TurnByC [80]</td>
<td>Rotate the attached orientation without changing the Cartesian coordinates.</td>
</tr>
<tr>
<td>Transform [81]</td>
<td>Place a PositionXY into the coordinate system referenced by this position.</td>
</tr>
</tbody>
</table>

### Required License

TC3 Physics 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>

### SetZero

Set the components of this vector to '0.0'.

**Syntax**

**Definition:**

```
METHOD SetZero
```

**Example: PositionXYC.SetZero**

**PLC declaration**

```
VAR
  position : PositionXYC;
VAR_END
```

**PLC implementation**

```
position.x := 0.3;
position.y := 0.5;
position.c := 20.0;
position.SetZero();
```

**Expected result**

```
position.x = 0
position.y = 0
position.c = 0
```

### SetValues

Set the components of this position.

**Syntax**

**Definition:**

```
```

---

TF5430  
Version: 1.0  
75
### SetValues

**Syntax**

```plaintext
METHOD SetValues
VAR_IN_OUT
  positionXYC : PositionXYC;
END_VAR
```

**Example: PositionXYC.SetValues**

**PLC declaration**

```plaintext
VAR
  position0 : PositionXYC;
  position1 : PositionXYC;
VAR_END
```

**PLC implementation**

```plaintext
position0.x := 5.6;
position0.y := 0.2;
position0.c := 4.2;
position1.SetValues (position0);
```

**Expected result**

- position1.x = 5.6
- position1.y = 0.2
- position1.c = 4.2

---

### SetValuesXY

**Syntax**

```plaintext
METHOD SetValuesXY
VAR_INPUT
  x : LREAL;
  y : LREAL;
END_VAR
```

**Example: PositionXYC.SetValuesXY**

**PLC declaration**

```plaintext
VAR
  position : PositionXYC;
  x : LREAL := 4.3;
  y : LREAL := 2.5;
VAR_END
```
PLC implementation

```plaintext
position.x := 0;
position.y := 0;
position.c := 3;
position.SetValuesXY (x, y);
```

Expected result

```plaintext
position.x = 4.3
position.y = 2.5
position.c = 3
```

**SetValuesXYC**

```plaintext
METHOD SetValuesXYC
VAR_INPUT
  x : LREAL;
  y : LREAL;
  c : LREAL;
END_VAR
```

Set the components of this position.

### Syntax

**Definition:**

```plaintext
METHOD SetValuesXYC
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>LREAL</td>
<td>x-component of the position.</td>
</tr>
<tr>
<td>y</td>
<td>LREAL</td>
<td>y-component of the position.</td>
</tr>
<tr>
<td>c</td>
<td>LREAL</td>
<td>c-component of the position scaled in degrees [°].</td>
</tr>
</tbody>
</table>

### Example: PositionXYC.SetValuesXYC

#### PLC declaration

```plaintext
VAR
  position : PositionXYC;
  x : LREAL := 4.3;
  y : LREAL := 2.5;
  c : LREAL := 20;
VAR_END
```

#### PLC implementation

```plaintext
position.x := 0;
position.y := 0;
position.c := 0;
position.SetValuesXYC (x, y, c);
```

#### Expected result

```plaintext
position.x = 4.3
position.y = 2.5
position.c = 20
```
**Invert**

Invert this position.

**Syntax**

**Definition:**

METHOD Invert

**Example: PositionXYC.Invert**

**PLC declaration**

```plc
VAR
    position : PositionXYC;
VAR_END
```

**PLC implementation**

```plc
position.SetValuesXYC (1, 3, 10);
position.Invert ();
```

**Expected result**

```plaintext
position.x = -1
position.y = -3
position.c = -10
```

**ConcatenateWith**

Multiply this position by a 2nd from the right.

**Syntax**

**Definition:**

METHOD ConcatenateWith

**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 to concatenate with.</td>
</tr>
</tbody>
</table>

**Example: PositionXYC.ConcatenateWith**

**PLC declaration**

```plc
VAR
    position0 : PositionXYC;
    position1 : PositionXYC;
VAR_END
```

**PLC implementation**

```plc
position0.SetValuesXYC (0.5, 1.2, 10);
position1.SetValuesXYC (0.3, 0.5, 15);
position0.ConcatenateWith (position1);
```
Expected result

position0.x = 0.7086
position0.y = 1.7450
position0.c = 25

**StepByLocalXY**

```
  dx  LREAL
  dy  LREAL
```

Apply a local step in respect to the local coordinate frame without changing the orientation.

Syntax

Definition:

METHOD StepByLocalXY
VAR_INPUT
  dx : LREAL;
  dy : LREAL;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx</td>
<td>LREAL</td>
<td>The displacement within the local x-direction.</td>
</tr>
<tr>
<td>dy</td>
<td>LREAL</td>
<td>The displacement within the local y-direction.</td>
</tr>
</tbody>
</table>

Example: PositionXYC.StepByLocalXY

PLC declaration

```
VAR
  position : PositionXYC;
  dx : LREAL := 2.0;
  dy : LREAL := 0.0;
VAR_END
```

PLC implementation

```
position.SetValuesXYC (0, 3, 10);
position.StepByLocalXY (dx, dy);
```

Expected result

position.x = 1.9696
position.y = 3.3473
position.c = 10

**ShiftByXY**

```
  dx  LREAL
  dy  LREAL
```

Displace the components of this position.

Syntax

Definition:
METHOD ShiftByXY
VAR_INPUT
   dx : LREAL;
   dy : LREAL;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx</td>
<td>LREAL</td>
<td>Shift of the x-component of this position.</td>
</tr>
<tr>
<td>dy</td>
<td>LREAL</td>
<td>Shift of the y-component of this position.</td>
</tr>
</tbody>
</table>

Example: PositionXYC.ShiftByXY

PLC declaration

VAR
   position : PositionXYC;
   dx : LREAL := 2.0;
   dy : LREAL := 0.0;
VAR_END

PLC implementation

position.SetValuesXYC (0, 3, 10);
position.ShiftByXY (dx, dy);

Expected result

dx = 2
dy = 3
position.c = 10

TurnByC

METHOD TurnByC
VAR_INPUT
   dc : LREAL;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc</td>
<td>LREAL</td>
<td>Angular displacement given in degrees [°].</td>
</tr>
</tbody>
</table>

Example: PositionXYC.TurnByC

PLC declaration

VAR
   position : PositionXYC;
   angle : LREAL := -5;
VAR_END

PLC implementation

position.SetValuesXYC (1.5, 2.5, 20);
position.TurnByC (angle);
Expected result
position.x = 1.5
position.y = 2.5
position.c = 15

**Transform**

Place a PositionXY into the coordinate system referenced by this position.

**Syntax**

**Definition:**

```plc
METHOD Transform
VAR_IN_OUT
    positionXY : PositionXY;
END_VAR
```

**In/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>positionXY</td>
<td>PositionXY</td>
<td>Position to transform.</td>
</tr>
</tbody>
</table>

**Example: PositionXYC.Transform**

**PLC declaration**

```plc
VAR
    position0 : PositionXYC;
    position1 : PositionXY;
VAR_END
```

**PLC implementation**

```plc
position0.SetValuesXYC (1, 2, 20);
position1.SetValuesXY (0, 2);
position0.Transform (position1);
```

**Expected result**

position1.x = 0.3160
position1.y = 3.8794

### 6.3 Tc3_Mc3PlanarMotion API

#### 6.3.1 Data Types

##### 6.3.1.1 Enums

**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>

6.3.1.1.2 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>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).
### 6.3.1.1.3 MC_SET_POSITION_MODE

Indicates the external setpoint generation mode.

**Syntax**

**Definition:**

```plaintext
TYPE MC_SET_POSITION_MODE :
{
    Absolute := 0,
    Relative := 1
}UINT;
END_TYPE
```

**Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>Indicates that the Planar Mover is following only the external setpoints.</td>
</tr>
<tr>
<td>Relative</td>
<td>Indicates that internal setpoint generation continues normally and the externally provided setpoints are added as an offset</td>
</tr>
</tbody>
</table>

### 6.3.1.2 Structs

### 6.3.1.2.1 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;
    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_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_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_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_COORDMODE</td>
<td>Mover coordinate mode information that is transferred from the Planar Mover to this function block.</td>
</tr>
</tbody>
</table>
### 6.3.1.2.2 CDT_PLCTOMC_PLANAR_MOVER

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

**Syntax**

**Definition:**
```plaintext
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_STD</td>
<td>Mover standard data that is transferred from this function block to the Planar Mover.</td>
</tr>
</tbody>
</table>

### 6.3.1.2.3 PlanarObjectInfo

Identifies a planar object uniquely by object id and type.

**Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>EPlanarObjectType [81]</td>
<td>Object type.</td>
</tr>
<tr>
<td>Id</td>
<td>UDINT</td>
<td>Object id.</td>
</tr>
</tbody>
</table>

### 6.3.1.2.4 ST_ExternalSetpointGenerationOptions

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

**Syntax**

**Definition:**
```plaintext
TYPE ST_ExternalSetpointGenerationOptions : STRUCT
  mode : MC_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_SET_POSITION_MODE [p. 83]</td>
<td>MC_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>

6.3.1.2.5  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>

6.3.1.2.6  ST_LeaveTrackOptions

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

Syntax

Definition:
```plaintext
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>

6.3.1.2.7  ST_MoveOnTrackOptions

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

Syntax

Definition:
6.3.1.2.8 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>

6.3.2 Function Blocks

6.3.2.1 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 [87]</td>
<td>Clears the Planar Environment (stators and boundary).</td>
</tr>
<tr>
<td>AddStator [87]</td>
<td>Adds a stator to the Planar Environment.</td>
</tr>
<tr>
<td>CreateBoundary [88]</td>
<td>Creates a boundary for the Planar Environment based on the previously added stator information or hardware information.</td>
</tr>
<tr>
<td>Update [88]</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
<tr>
<td>AddToGroup [88]</td>
<td>Adds the Planar Environment to the given Planar Group.</td>
</tr>
<tr>
<td>RemoveFromGroup [89]</td>
<td>Removes the Planar Environment from its current Planar Group, i.e. disables collision checks.</td>
</tr>
</tbody>
</table>
### GetPlanarObjectInfo

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPlanarObjectInfo</td>
<td>Returns environment object info (type: environment, id: OID of nc</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>

### 6.3.2.1.1 Clear

```plaintext
Clear

commandFeedback : MC_PlanarFeedback
```

Clears the Planar Environment (stators and boundary).

#### Syntax

**Definition:**

```plaintext
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>

### 6.3.2.1.2 AddStator

```plaintext
AddStator

commandFeedback : MC_PlanarFeedback
lowerX : LREAL
lowerY : LREAL
```

Adds a stator to the Planar Environment.

#### Syntax

**Definition:**

```plaintext
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>commandFeed back</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>

### 6.3.2.1.3 CreateBoundary

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

**Syntax**

Definition:

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

### 6.3.2.1.4 Update

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

**Syntax**

Definition:

```plaintext
METHOD Update
```

### 6.3.2.1.5 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>

### 6.3.2.1.6 RemoveFromGroup

**RemoveFromGroup**

```
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>

### 6.3.2.1.7 GetPlanarObjectInfo

**GetPlanarObjectInfo**

```
PlanarObjectInfo
```

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

**Syntax**

Definition:

```plaintext
METHOD GetPlanarObjectInfo : PlanarObjectInfo
```
### Return value

PlanarObjectInfo [\(^84\)]

### 6.3.2.2 MC_PlanarFeedback

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

#### Syntax

**Definition:**

```plaintext
FUNCTION_BLOCK MC_PlanarFeedback
VAR_OUTPUT
  objectInfo : PlanarObjectInfo;
  Active : BOOL;
  Busy : BOOL;
  Done : BOOL;
  Aborted : BOOL;
  Error : BOOL;
  ErrorId : UDINT;
END_VAR
```

#### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectInfo</td>
<td>PlanarObjectInfo [(^84)]</td>
<td>Indicates which object one would collide with.</td>
</tr>
<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>
</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>

### 6.3.2.2.1 Update

Updates internal state of the object.

**Syntax**

**Definition:**

`METHOD Update`

### 6.3.2.3 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 [91]</td>
<td>Starts enabling the Planar Group.</td>
</tr>
<tr>
<td>Disable [92]</td>
<td>Starts disabling the Planar Group.</td>
</tr>
<tr>
<td>Reset [92]</td>
<td>Starts resetting the Planar Group.</td>
</tr>
<tr>
<td>Update [93]</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>

### 6.3.2.3.1 Enable

**Syntax**

**Definition:**

Starts enabling the Planar Group.
PLC Libraries

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>

6.3.2.3.2 Disable

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>

6.3.2.3.3 Reset

Starts resetting the Planar Group.

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>
6.3.2.3.4 Update

**Update**

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

**Syntax**

**Definition:**

```
METHOD Update
```

### 6.3.2.4 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveToPosition</td>
<td>Initiates a direct movement to the specified position.</td>
</tr>
<tr>
<td>JoinTrack</td>
<td>Initiates a direct movement to the specified track. At the end of the</td>
</tr>
<tr>
<td></td>
<td>movement the mover joins the track.</td>
</tr>
<tr>
<td>LeaveTrack</td>
<td>Initiates a direct movement to the specified position. At the beginning of</td>
</tr>
<tr>
<td></td>
<td>the movement the track is left.</td>
</tr>
<tr>
<td>MoveOnTrack</td>
<td>Initiates a movement on the track to the specified position and returns</td>
</tr>
<tr>
<td></td>
<td>command ID.</td>
</tr>
<tr>
<td>MoveZ</td>
<td>Initiates a movement for the z component.</td>
</tr>
<tr>
<td>MoveA</td>
<td>Initiates a movement for the a component.</td>
</tr>
<tr>
<td>MoveB</td>
<td>Initiates a movement for the b component.</td>
</tr>
<tr>
<td>MoveC</td>
<td>Initiates a movement for the c component.</td>
</tr>
<tr>
<td>Halt</td>
<td>Initiates a halt.</td>
</tr>
<tr>
<td>Enable</td>
<td>Starts enabling the Planar Mover.</td>
</tr>
<tr>
<td>Disable</td>
<td>Starts disabling the Planar Mover.</td>
</tr>
<tr>
<td>Reset</td>
<td>Starts resetting the Planar Mover.</td>
</tr>
<tr>
<td>Update</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
<tr>
<td>SetPosition</td>
<td>Sets the position of the Planar Mover. Only possible if the Planar Mover is</td>
</tr>
<tr>
<td></td>
<td>disabled</td>
</tr>
<tr>
<td>StartExternalSetpointGeneration</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</td>
<td>Ends the external setpoint generation, called after last SetExternalSetpoint (in the same PLC cycle).</td>
</tr>
<tr>
<td>SetExternalSetpoint</td>
<td>Sets the external setpoint for the Planar Mover, must be called each PLC</td>
</tr>
<tr>
<td></td>
<td>cycle during external setpoint generation.</td>
</tr>
<tr>
<td>AddToGroup</td>
<td>Adds the Planar Mover to the given Planar Group.</td>
</tr>
<tr>
<td>RemoveFromGroup</td>
<td>Removes the Planar Mover from its current Planar Group, i.e. disables</td>
</tr>
<tr>
<td></td>
<td>collision checks.</td>
</tr>
<tr>
<td>GetPlanarObjectInfo</td>
<td>Returns mover object info (type: mover, id: OID of nc mover).</td>
</tr>
</tbody>
</table>
PLC Libraries

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>

6.3.2.4.1 MoveToPosition

**MoveToPosition**

- **commandFeedback**: MC_PlanarFeedback
- **constraint**: DynamicConstraint_Path
- **options**: ST_MoveToPositionOptions
- **targetPosition**: PositionXYC

Initiates a direct movement to the specified position.

**Syntax**

**Definition:**

```plaintext
METHOD MoveToPosition
  VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    constraint : DynamicConstraint_Path;
    options : ST_MoveToPositionOptions;
  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 [90]</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
<tr>
<td>options</td>
<td>ST_MoveToPositionOptions [86]</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>
6.3.2.4.2 JoinTrack

**JoinTrack**

- `commandFeedback` : `MC_PlanarFeedback`
- `constraint` : `DynamicConstraint_Path`
- `options` : `ST_JoinTrackOptions`
- `targetTrack` : `MC_PlanarTrack`

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       : DynamicConstraint_Path;
  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><code>MC_PlanarFeedback</code></td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>constraint</td>
<td><code>DynamicConstraint_Path</code></td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
<tr>
<td>options</td>
<td><code>ST_JoinTrackOptions</code></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><code>MC_PlanarTrack</code></td>
<td>Target track for the movement.</td>
</tr>
</tbody>
</table>

6.3.2.4.3 LeaveTrack

**LeaveTrack**

- `commandFeedback` : `MC_PlanarFeedback`
- `constraint` : `DynamicConstraint_Path`
- `options` : `ST_LeaveTrackOptions`
- `targetPosition` : `PositionXYZ`

Initiates a direct movement to the specified position. At the beginning of the movement the track is left.

**Syntax**

**Definition:**

```plaintext
METHOD LeaveTrack
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  constraint       : DynamicConstraint_Path;
  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>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</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>

6.3.2.4.4 MoveOnTrack

```plaintext
MoveOnTrack
  commandFeedback  : MC_PlanarFeedback
  targetTrack      : MC_PlanarTrack
  targetPositionOnTrack : LREAL
  constraint       : DynamicConstraint_Path
  options          : ST_MoveOnTrackOptions
```

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

Syntax

Definition:

```plaintext
METHOD MoveOnTrack
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  targetTrack     : MC_PlanarTrack;
  targetPositionOnTrack : LREAL;
  constraint      : DynamicConstraint_Path;
  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</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_Path</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>
6.3.2.4.5 MoveZ

```plaintext
MoveZ

commandFeedback : MC_PlanarFeedback
targetPosition : LREAL
constraint : DynamicConstraint_Path
```

Initiates a movement for the z component.

### Syntax

**Definition:**

```plaintext
METHOD MoveZ
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  targetPosition : LREAL;
  constraint : DynamicConstraint_Path;
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>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
</tbody>
</table>

6.3.2.4.6 MoveA

```plaintext
MoveA

commandFeedback : MC_PlanarFeedback
targetPosition : LREAL
constraint : DynamicConstraint_Path
```

Initiates a movement for the a component.

### Syntax

**Definition:**

```plaintext
METHOD MoveA
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
  targetPosition : LREAL;
  constraint : DynamicConstraint_Path;
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>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
</tbody>
</table>
6.3.2.4.7 MoveB

<table>
<thead>
<tr>
<th>METHOD MoveB</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>commandFeedback : MC_PlanarFeedback;</td>
</tr>
<tr>
<td>targetPosition : LREAL;</td>
</tr>
<tr>
<td>constraint : DynamicConstraint_Path;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

Initiates a movement for the b component.

**Syntax**

Definition:

METHOD MoveB
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    targetPosition : LREAL;
    constraint : DynamicConstraint_Path;
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 [90]</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>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
</tbody>
</table>

6.3.2.4.8 MoveC

<table>
<thead>
<tr>
<th>METHOD MoveC</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>commandFeedback : MC_PlanarFeedback;</td>
</tr>
<tr>
<td>targetPosition : LREAL;</td>
</tr>
<tr>
<td>constraint : DynamicConstraint_Path;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

Initiates a movement for the c component.

**Syntax**

Definition:

METHOD MoveC
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    targetPosition : LREAL;
    constraint : DynamicConstraint_Path;
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 [90]</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>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (V,A,D,J).</td>
</tr>
</tbody>
</table>
6.3.2.4.9 Halt

**Halt**

```plaintext
commandFeedback : MC_PlanarFeedback
constraint : DynamicConstraint_Path
```

Initiates a halt.

**Syntax**

**Definition:**

```plaintext
METHOD Halt
VAR_INPUT
    commandFeedback : MC_PlanarFeedback;
    constraint : DynamicConstraint_Path;
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>DynamicConstraint_Path</td>
<td>Constraint on maximal dynamics during the movement (D,J).</td>
</tr>
</tbody>
</table>

6.3.2.4.10 Enable

**Enable**

```plaintext
commandFeedback : MC_PlanarFeedback
```

Starts enabling the Planar Mover.

**Syntax**

**Definition:**

```plaintext
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>

6.3.2.4.11 Disable

**Disable**

```plaintext
commandFeedback : MC_PlanarFeedback
```

Starts disabling the Planar Mover.

**Syntax**

**Definition:**

```plaintext
```
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>

6.3.2.4.12 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>

6.3.2.4.13 Update

Update

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

Syntax

Definition:
METHOD Update

6.3.2.4.14 SetPosition

SetPosition

commandFeedback : MC_PlanarFeedback
position : PositionXYC

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

Syntax

Definition:
PLC Libraries

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>
<tbody>
<tr>
<td>commandFeedback</td>
<td>MC_PlanarFeedback [90]</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>New position of the Planar Mover.</td>
</tr>
</tbody>
</table>

6.3.2.4.15 StartExternalSetpointGeneration

```plaintext
StartExternalSetpointGeneration
commandFeedback  MC_PlanarFeedback
options          ST_ExternalSetpointGenerationOptions
```

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

### Syntax

**Definition:**

```plaintext
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>commandFeedback</td>
<td>MC_PlanarFeedback [90]</td>
<td>The feedback object for the command.</td>
</tr>
<tr>
<td>options</td>
<td>ST_ExternalSetpointGenerationOptions [84]</td>
<td>Options for the movement.</td>
</tr>
</tbody>
</table>

6.3.2.4.16 StopExternalSetpointGeneration

```plaintext
StopExternalSetpointGeneration
commandFeedback  MC_PlanarFeedback
```

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

### Syntax

**Definition:**

```plaintext
METHOD StopExternalSetpointGeneration
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR
```
PLC Libraries

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>commandFeedback</td>
<td>MC_PlanarFeedback</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

### 6.3.2.4.17 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>setAcceleration</td>
<td>MoverVector</td>
<td>Acceleration that is send to the Planar Mover.</td>
</tr>
</tbody>
</table>

### 6.3.2.4.18 AddToGroup

**AddToGroup**

- commandFeedback  : MC_PlanarFeedback
- group : MC_PlanarGroup

Adds the Planar Mover to the given Planar Group.

**Syntax**

**Definition:**

METHOD AddToGroup
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR
VAR_IN_OUT
  group : MC_PlanarGroup;
END_VAR
6.3.2.4.19 RemoveFromGroup

**Definition:**

METHOD RemoveFromGroup
VAR_INPUT
  commandFeedback : MC_PlanarFeedback;
END_VAR

**Syntax**

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

**Return value**

PlanarObjectInfo [84]
6.3.2.5 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[104]</td>
<td>Clears the geometric information of the Planar Track.</td>
</tr>
<tr>
<td>AppendPosition[105]</td>
<td>Appends a position to the Planar Track.</td>
</tr>
<tr>
<td>AppendLine[105]</td>
<td>Appends a line to the Planar Track.</td>
</tr>
<tr>
<td>AppendCircle[106]</td>
<td>Appends a circular arc to the Planar Track.</td>
</tr>
<tr>
<td>CloseLoop[107]</td>
<td>Closes the loop of the Planar Track, no further part can be appended.</td>
</tr>
<tr>
<td>StartFromTrack[107]</td>
<td>Sets the other Planar Tracks endpoint as start point of this Planar Track,</td>
</tr>
<tr>
<td></td>
<td>transition is smooth. The other Planar Track is blocked for further changes</td>
</tr>
<tr>
<td></td>
<td>(until it is cleared).</td>
</tr>
<tr>
<td>EndAtTrack[108]</td>
<td>Appends a smooth transition from the end of this Planar Track to the other</td>
</tr>
<tr>
<td></td>
<td>Planars start point. The Planar Track is blocked for further changes</td>
</tr>
<tr>
<td></td>
<td>(until it is cleared).</td>
</tr>
<tr>
<td>Enable[108]</td>
<td>Starts enabling the Planar Track.</td>
</tr>
<tr>
<td>Disable[109]</td>
<td>Starts disabling the Planar Track.</td>
</tr>
<tr>
<td>Reset[109]</td>
<td>Starts resetting the Planar Track.</td>
</tr>
<tr>
<td>GetArcLengthClosestTo[109]</td>
<td>Calculate the arc length value where the Planar Track is closest to a</td>
</tr>
<tr>
<td></td>
<td>geometry's center point.</td>
</tr>
<tr>
<td>GetPositionAt[110]</td>
<td>Get a position on the Planar Track at a specific arc length value.</td>
</tr>
<tr>
<td>GetLength[110]</td>
<td>Returns the Planar Tracks length, -1 return value indicates no connection</td>
</tr>
<tr>
<td></td>
<td>to Nc Track.</td>
</tr>
<tr>
<td>Update[111]</td>
<td>Updates internal state of the object, must be triggered each cycle.</td>
</tr>
<tr>
<td>AddToGroup[111]</td>
<td>Adds the Planar Track to the given Planar Group.</td>
</tr>
<tr>
<td>RemoveFromGroup[112]</td>
<td>Removes the Planar Track from its current Planar Group, i.e. disables</td>
</tr>
<tr>
<td></td>
<td>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>

6.3.2.5.1 Clear

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 [90]</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

6.3.2.5.2 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 [90]</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>

6.3.2.5.3 AppendLine

AppendLine

commandFeedback    MC_PlanarFeedback
start               PositionXYC
end                 PositionXYC

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>

### 6.3.2.5.4 AppendCircle

**AppendCircle**

- commandFeedback : MC_PlanarFeedback
- clockwise : BOOL
- start : PositionXYC
- end : PositionXYC
- center : PositionXY

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>
</tbody>
</table>
### 6.3.2.5.5 CloseLoop

**CloseLoop**

```plaintext
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

<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>

### 6.3.2.5.6 StartFromTrack

**StartFromTrack**

```plaintext
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

<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>track</td>
<td>MC_PlanarTrack</td>
<td>The other Planar Track.</td>
</tr>
</tbody>
</table>

#### 6.3.2.5.7 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>
<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>track</td>
<td>MC_PlanarTrack</td>
<td>The other Planar Track.</td>
</tr>
</tbody>
</table>

#### 6.3.2.5.8 Enable

```plaintext
Enable
commandFeedback MC_PlanarFeedback
```

Starts enabling the Planar Track.

**Syntax**

**Definition:**

```plaintext
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 [► 90]</td>
<td>The feedback object for the command.</td>
</tr>
</tbody>
</table>

#### 6.3.2.5.9 Disable

**Disable**

starts disabling the planar track.

**Syntax**

Definition:

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

#### 6.3.2.5.10 Reset

**Reset**

Starts resetting the Planar Track.

**Syntax**

Definition:

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

#### 6.3.2.5.11 GetArcLengthClosestTo

**GetArcLengthClosestTo**

starts getting the arc length closest to the geometry.
Calculate the arc length value where the Planar Track is closest to a geometry's center point.

**Syntax**

**Definition:**

```plaintext
METHOD GetArcLengthClosestTo : LREAL
VAR_IN_OUT
   geometry : IPlcGeometry2D;
END_VAR
```

<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

### 6.3.2.5.12 GetPositionAt

```plaintext
GetPositionAt
```

- **arcLength** : LREAL
- **position** : PositionXYC

Get a position on the Planar Track at a specific arc length value.

**Syntax**

**Definition:**

```plaintext
METHOD GetPositionAt
VAR_INPUT
   arcLength : LREAL;
END_VAR
VAR_IN_OUT
   position : PositionXYC;
END_VAR
```

<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>

<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>

### 6.3.2.5.13 GetLength

```
GetLength
```

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

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

Syntax

Definition:
METHOD GetPlanarObjectInfo : PlanarObjectInfo

Return value

PlanarObjectInfo [84]

6.3.2.5.15 Update

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

Syntax

Definition:
METHOD Update

6.3.2.5.16 AddToGroup

Adds the Planar Track 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>commandFeed</td>
<td>MC_PlanarFeedback [90]</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 [91]</td>
<td>The Planar Group that the mover joins.</td>
</tr>
</tbody>
</table>

### 6.3.2.5.17 RemoveFromGroup

**Syntax**

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

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