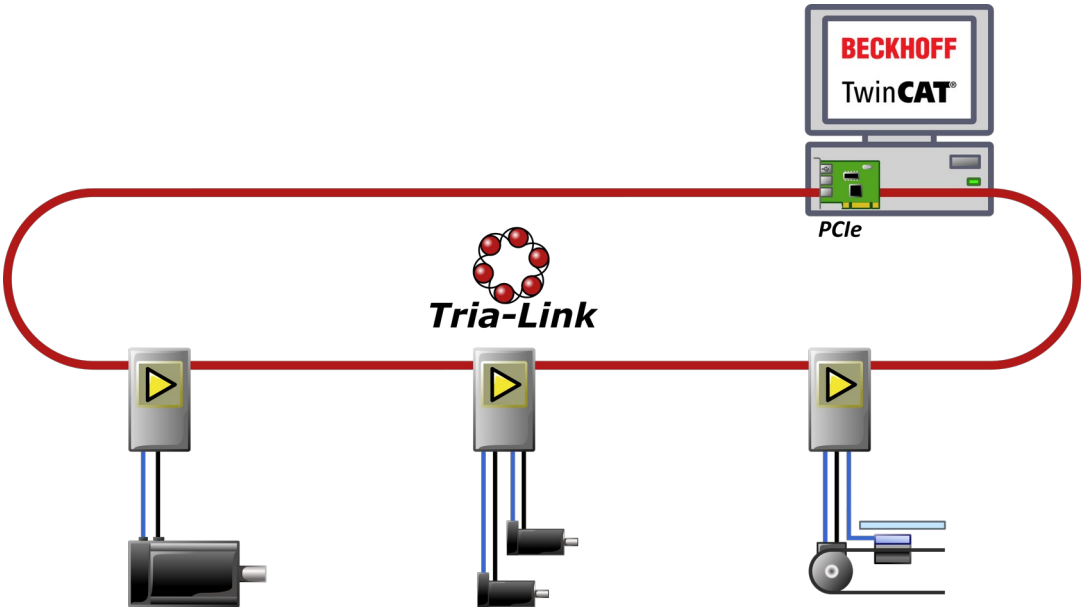
TwinCAT Setup Guide

Tria-Link Fieldbus

This setup guide explains, how to setup *Beckhoff TwinCAT* in conjunction with *Triamec* drives and the *Tria-Link* fieldbus.
This guide assumes that the user is familiar with *Beckhoff TwinCAT* and its dependencies.



Document	SWTC_TwinCAT-UserGuide_EP
Version	032, 2025-01-13
Source	Q:\Doc\Software\SWTC\SWTC_TwinCAT-UserGuide\
Destination	T:\doc\Software
Owner	rb

www.triamec.com

Table of Contents

1 Configuring the Drives

1.1 TAM System Explorer

To configure the drives use the *TAM System Explorer*.

Use USB or Ethernet to connect to a drive or to the whole *Tria-Link bus*. Ensure that the *TAM System Explorer* does not use the PCIe interface. Therefore, check the setting in **File > Preferences > Acquired Adapters** (Figure 1). Refer to [1] on further details regarding connectivity.

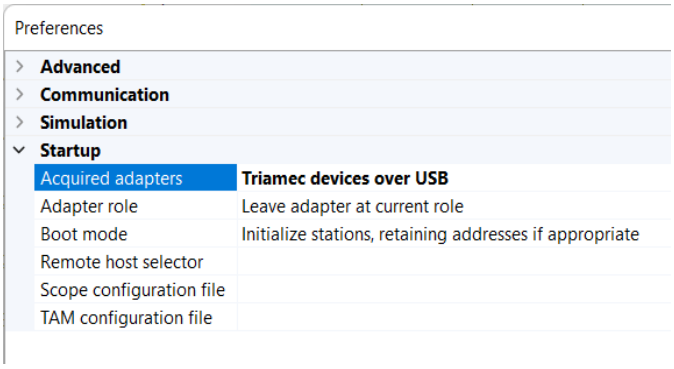


Figure 1: Adapter setting in the TAM System Explorer

1.2 Setting Up the Drives

Before a drive can be used with *TwinCAT*, it must be configured with the *TAM System Explorer*. Consult [7] to know more about physical media and connection of *Tria-Link* drives. Refer to [1] on how to setup parameters, install new firmware and saving parameters persistently on the drive.

With *Tria-Link*, drives are accessed using their unique station address. If used with *TwinCAT* this address has to be assigned manually for each drive by applying the following steps. The allowed address range is 1 to 48.

- Use the *TAM System Explorer* to set the drive address at General.Parameters.LinkAddress.
- Set General.Parameters.UseDedicatedLinkAddresses to True.
- To prevent loss of drive settings after a power cycle the configuration has to be saved persistently on the drive as described in [1]. Later changes of the configuration need to be persisted again.
- Also save the configuration as a `*.TAMcfg` backup file on the PC.

Note It's recommended to set up the axes in the same units as later used on the *TwinCAT* side (i.e. mm or degrees).

2 TwinCAT Setup

The following prerequisites are mandatory to start integrating *Triamec* drives into *TwinCAT* solutions. Follow subsequent instructions independent of working on a new solution or integrate into existing solutions.

- A PC with a *Tria-Link* adapter (*TL*) installed.
- An installation of *TwinCAT*.
- Additional *TwinCAT Functions (TFxxx)* mentioned in this guide (i.e. *CNC* or *NCI*).
- Setup your drives using the *TAM System Explorer* as explained in chapter 1.
- All drives must be powered and connected to the *Tria-Link* adapter building a ring topology.
- The *Release Package* `PLC_Release<Version>.zip`, available as download from the *Triamec* website [6], *Tria-Link* section.

At this point, open your *TwinCAT* solution, or create a new one.

Note For the following steps, *TwinCAT* must be in configuration mode.

2.1 Tria-Link Adapter

To add a *Tria-Link* Adapter to the *TwinCAT* solution.

1. Select **Add New Item…** in the context menu of the **I/O > Devices** node.
2. In the dialog select **Miscellaneous > Generic NOV-DP-RAM** (Figure 2).
   For the purpose of this guide the device is named *Trialink*.

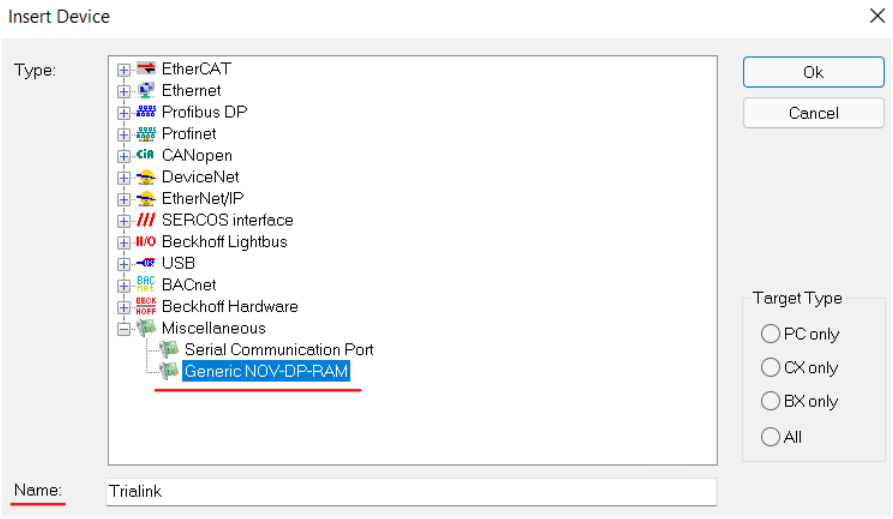


Figure 2: Add a Tria-Link Adapter to the solution

3. Open **I/O > Devices > Trialink > Generic NOV-DP-RAM Device** (Figure 3).
4. Set the **Vendor ID** to 1618 and the **Device ID** to 0211 and press **Search...** to find the Adapter.

Note The device *TL* (0x211) requires a DMA driver for optimal performance. This is installed automatically with the *TAM Software*. If the *TAM Software* is not installed on this PC, install the driver manually from the folder `dmaDrivers` in the *Release Package*.

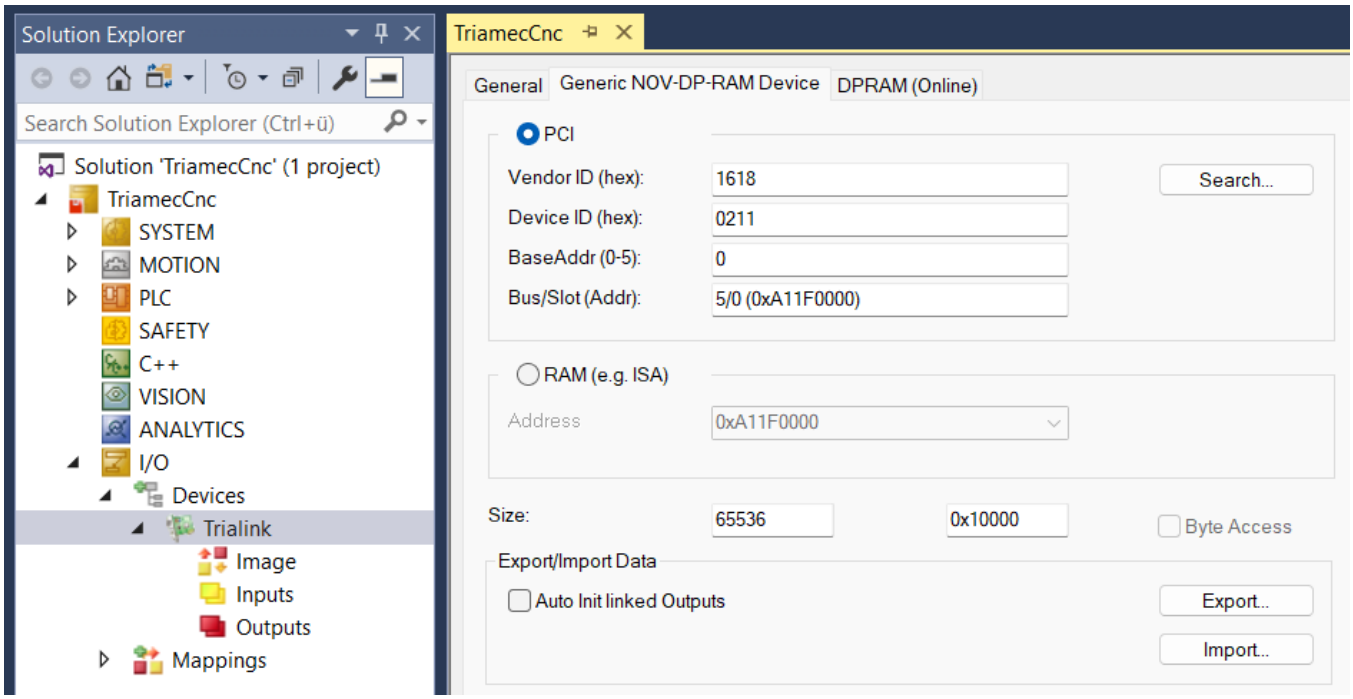


Figure 3: Tria-Link Adapter - General DPRAM settings.

2.2 Tasks and Programs

It is mandatory to split calls to the *Triamec* library into two tasks. From here on we talk about the *TASK_SLOW* and the *TASK_FAST*. Create the tasks by right-clicking the *PLC* project and select **Add > Referenced Task...**.

Corresponding to the tasks two *PLC programs (PRG)* *MAIN_SLOW* and *MAIN_FAST* have to be introduced.

The programs are linked to the tasks respectively, by dragging the *PRG* onto the task (Figure 4).

In the *MAIN_SLOW (PRG)* configurations, states and low priority values are updated. A `CallSlow()` method is called for each object.

```
TL_Trialink2.CallSlow();
TL_Axis2.CallSlow();
```

The *MAIN_FAST (PRG)* is mainly reserved to update critical data where a high update rate is required, such as position data.
For the fast cycle `CallFast()` methods are called on the `TL_Axis2` and `TL_Trialink2` objects.

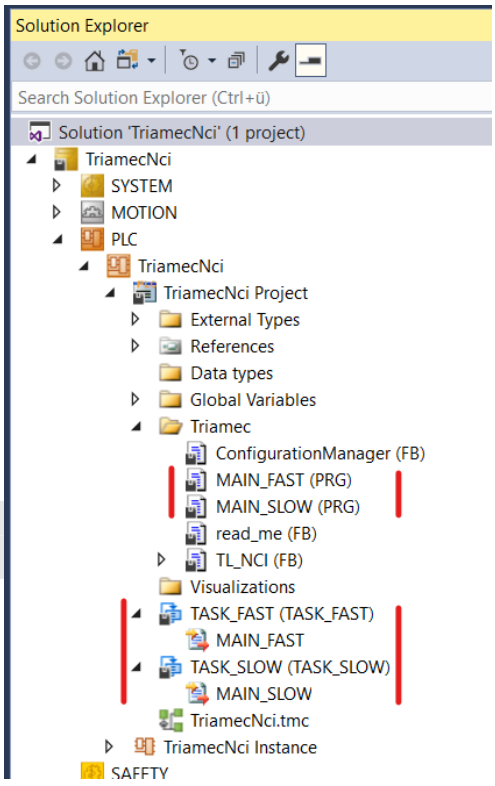


Figure 4: PLC Tasks and linked PRGs

```
TL_Trialink2.CallFast();
TL_Axis2.CallFast();
```

Further information on the *PLC* code is described in chapter 6.

2.3 Install the Triamec Library

Note If you start with a new solution you may first add a new *PLC Project* to then reference the `Triamec` library.

The library is shipped with the *Release Package,* named `TriamecLib<version>.compiled-library`.
Installation:

1. In *TwinCAT*, go to **PLC > Library Repository…** and **Install** the *TriamecLib* by browsing to the file mentioned above.
2. Add it to your PLC project through the context menu **Add library...** of the **References** folder within the PLC project node of your solution.
3. Also reference the *TC3_Module* library if not already present.

2.4 Configuration

The following assignments are mandatory in the PLC code (see also chapter 2.2).

Assign the device id to the `TL_Trialink2` instance in the configuration code of the *PLC* project. The id is found at **I/O > Devices > Trialink > General > Id.**

```
Trialink.Config.nDevId := 1; // NOV-DP-RAM Id, see TwinCAT System Manager (I/0 > Devices > TriaLink)
```

Axis configuration is mandatory on a `TL_Axis2` instance. Set at least the following parameters.

Parameter	Value
`TL_Axis2.Config.Simulate`	`TRUE` or `FALSE`, for real or virtual axis respectively.
`TL_Axis2.Config.Station`	Station address of the drive as set in chapter 1.2. For dual channel drives (TSD), set the same station address for both axes.
`TL_Axis2.Config.SubAxis`	`TL_Config.SubAxis.FirstAxis` for Axis0, `TL_Config.SubAxis.SecondAxis` for Axis1 (*TSD* drives).
`TL_Axis2.Config.GearFactor`	Default `1`; Multiplication factor of position and velocity data. Change this value to scale in between units of PLC and drive (see also Figure 5).
`TL_Axis2.Config.ModuloWrap`	Modulo wrap for rotational axes, see also chapter 8.1.
`TL_Axis2.iAxis`	Default 0; Axes should be uniquely labeled between 1-32.

2.5 Position Scaling

TwinCAT uses integer (32bit) for positions whereas *Triamec* drives use the double (64bit) format. Therefore, the code needs to convert the integer coming from the NC to the double value used in the *PLC* code, and v.v.

The standard conversion value between *CNC*/*NCI* and *PLC* is 10'000 inc/unit. Refer to chapter 3.5 for *NCI* and chapter 4.6 for *CNC* setups on how to increase the resolution.

The conversions must be considered when exchanging position data. Take a look at the examples (chapter 5), where the conversion contains the scale variable `inc_per_unit`.

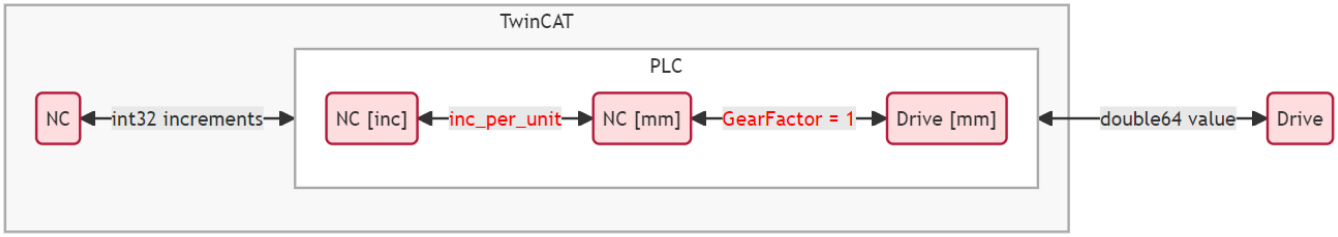


Figure 5: Unit conversions of position data in PLC code

Note We recommend to set up the drive in the same position units as used in the application to remove complexity. `GearFactor = 1` is therefore the default value.

2.6 Real-Time configuration

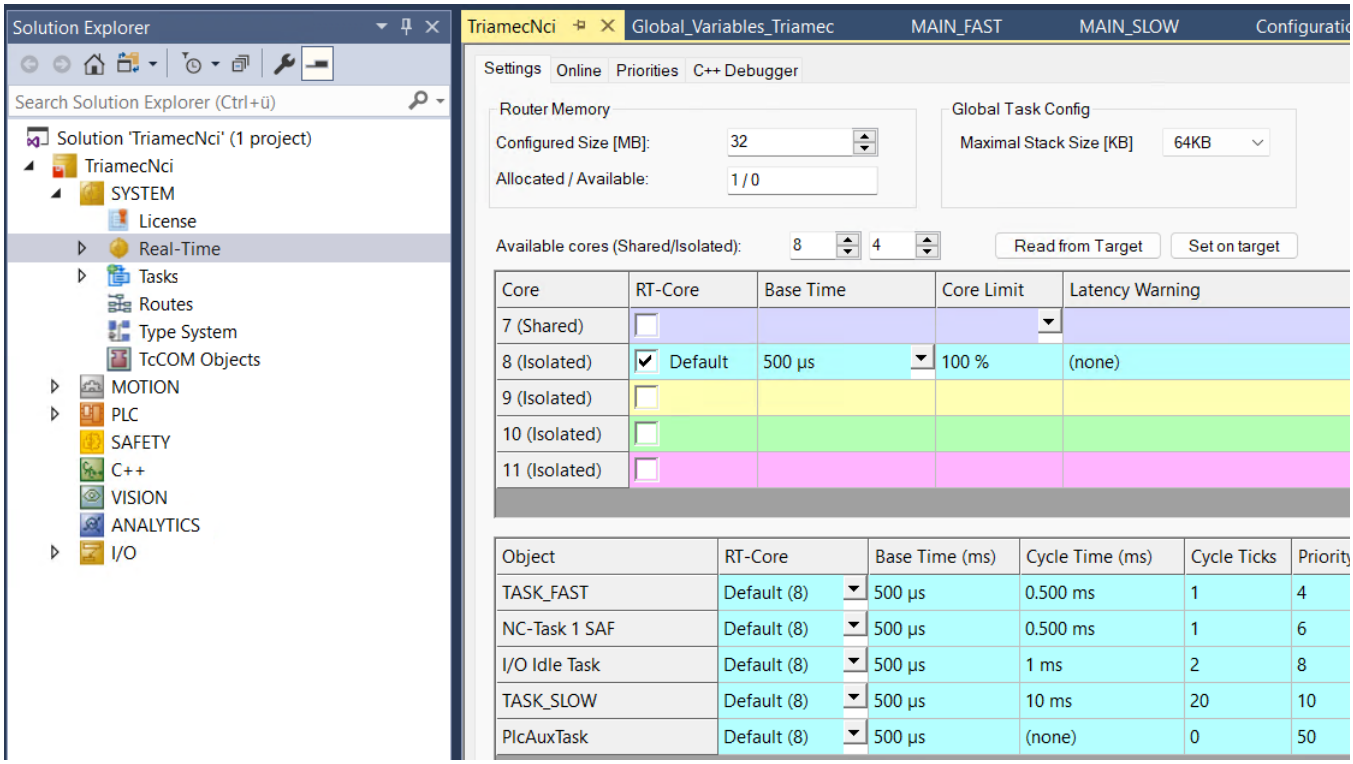


Figure 6: System Real-Time configuration

Open the node **SYSTEM > Real-Time** and click **Read from Target** to read the **Available cores**. *Triamec* recommends to have at least one isolated core (Figure 6). The *TASK_FAST* and the *TASK_SLOW* must run on the same **Core**.

From here the setup using *NCI* differs from using *CNC*. According to your setup proceed in chapter 3 for *NCI* or chapter 4 for *CNC*.

3 NCI Setup

For this chapter it's assumed that chapter 2 has been worked through.

3.1 NCI Axes

Add the **NCI Configuration** by selecting **Add new Item …** in the context menu of the **MOTION** node. Then find the **Axes** node in the generated **NC-Task** and add the amount of axes matching your setup by selecting **Add new Item …** in the context menu of the **Axes** node. Select the Axis type **Continuous Axis** in the dialog (Figure 7).

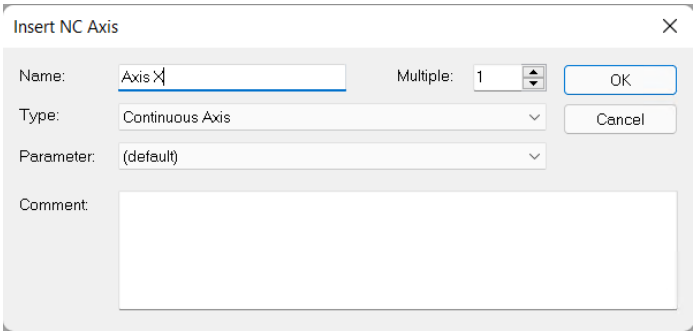


Figure 7: Insert NC Axis dialog

Now go through each axis **Settings** and set the **Axis Type** to **SERCOS Drive** (Figure 8).

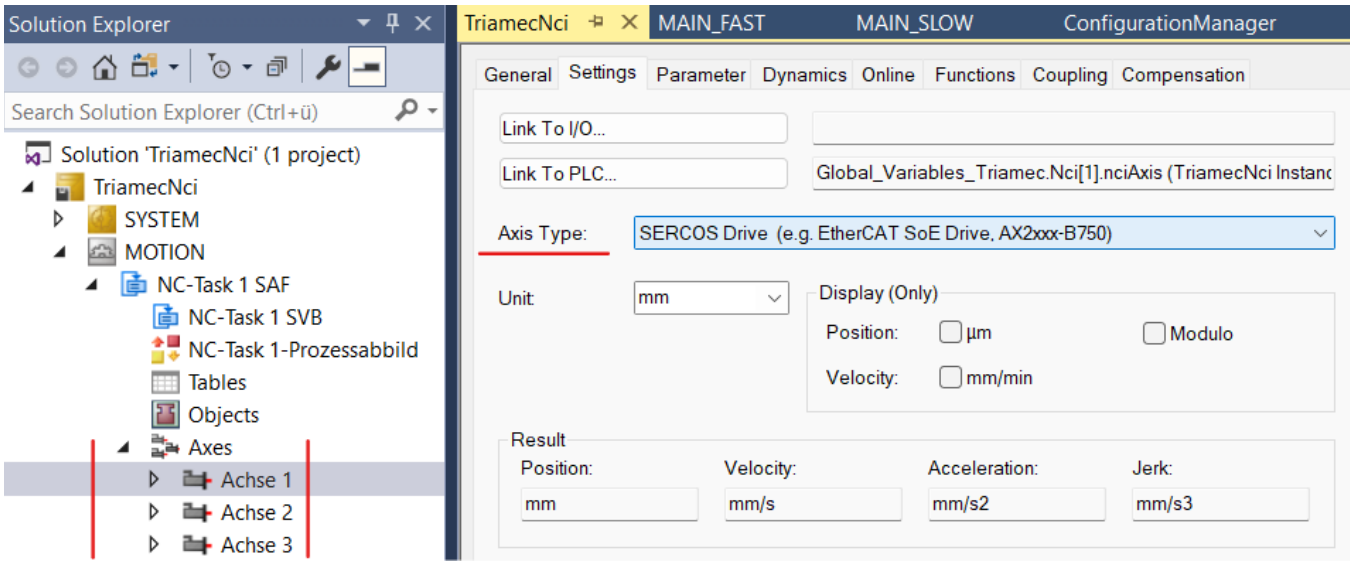


Figure 8: Axis Type configuration

3.2 NCI Tasks

The *TASK_FAST* must run at the same cycle time as the *NC-Task SAF* because it receives the path planner data from the *SAF* task and forwards it to the *Tria-Link* and vice versa.

To achieve this, right-click the **PLC > Project > TASK_FAST**, select **Assign to task** and choose the **NC-Task SAF**. This makes the *TASK_FAST* obsolete, which can be removed under **SYSTEM > Tasks**.

It is strongly recommended to give the NC task the highest priority and set its cycle rate to one tick. Both settings can be changed in **MOTION > NC-Task SAF > Task**.

An overview of the task cycle rates and priorities is available in the **SYSTEM > Real-Time** settings (see also chapter 2.6).

3.3 Linking Axes and PLC

Because the interface to the drive data is on PLC level, we need to make some process variables to link to the NC level. The process variables are also categorized into slow and fast data and must be used with the corresponding tasks.

The general axis interface becomes available by making one instance of *AXIS_REF* for each axis.

Note *AXIS_REF (FB)* is available from the *Beckhoff* library *Tc2_MC2*. Therefore, the library has to be referenced in the *PLC* project.

- For each `AXIS_REF` instance, an **Inputs > NcToPlc** variable is generated.
  Link these to the corresponding *NCI* axes output with suffix **ToPlc**.
- For each `AXIS_REF` instance, an **Outputs > PlcToNc** is generated.
  Link these to the corresponding NCI axes output with suffix **FromPlc**.

3.3.1 SERCOS Interface (Slow Task)

Two additional process variables per axis are needed for the NC to get control on the drive state.

```
State1 AT %Q* : BYTE;  (*link to: NC > Axis > Drive > Inputs > nState1 *)
State4 AT %Q* : BYTE;  (*link to: NC > Axis > Drive > Inputs > nState4 *)
```

Basic state handling code is available in the Triamec examples on *GitHub*, see chapter 5.

3.3.2 Position Data Exchange (Fast Task)

Two additional process variables per axis are made to exchange position data in the fast task.

```
PosCmd AT %I* : DWORD;  (*link to: NC > Axis > Drive > Outputs > nDataOut1 *)
PosAct AT %Q* : DWORD;  (*link to: NC > Axis > Enc > Inputs > nDataIn1 *)
```

In the building process an instance of your *PLC* project is generated with the **Inputs** and **Outputs**. Use the context menu entry **Change Link …** on the variables to setup the links above (Figure 9). Check that no other links are active by clearing them via **context menu > Clear Link(s)**.

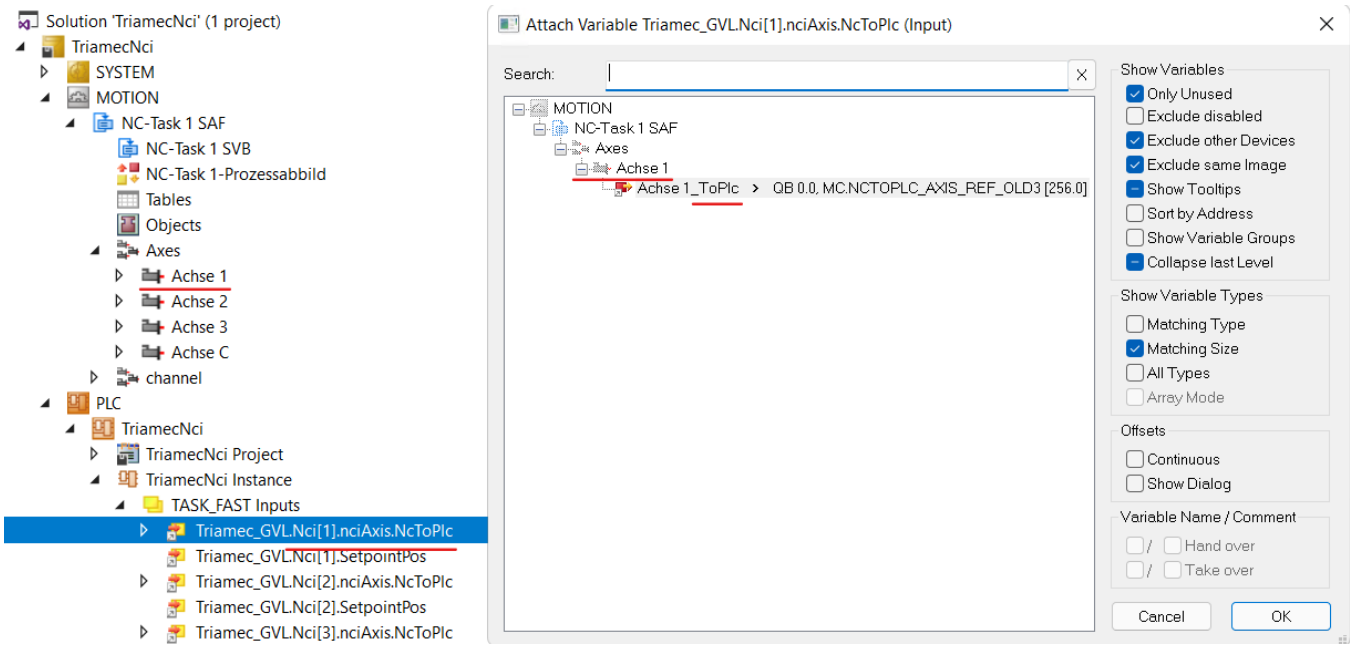


Figure 9: Link PLC instance to NCI

3.4 Encoder Mask

Change the following parameters at **MOTION > Axes > Axis N > Enc > Parameter** for each axis in the setup.

- **Encoder Evaluation > Encoder Mask** = 0xFFFFFFFF
- **Encoder Evaluation > Encoder Sub Mask** = 0xFFFFFFFF

3.5 Change Resolution

To change the resolution of an axis, first change the following parameter at **MOTION > Axes > Axis N > Enc > Parameter > Scaling Factor Denominator** (Figure 10).

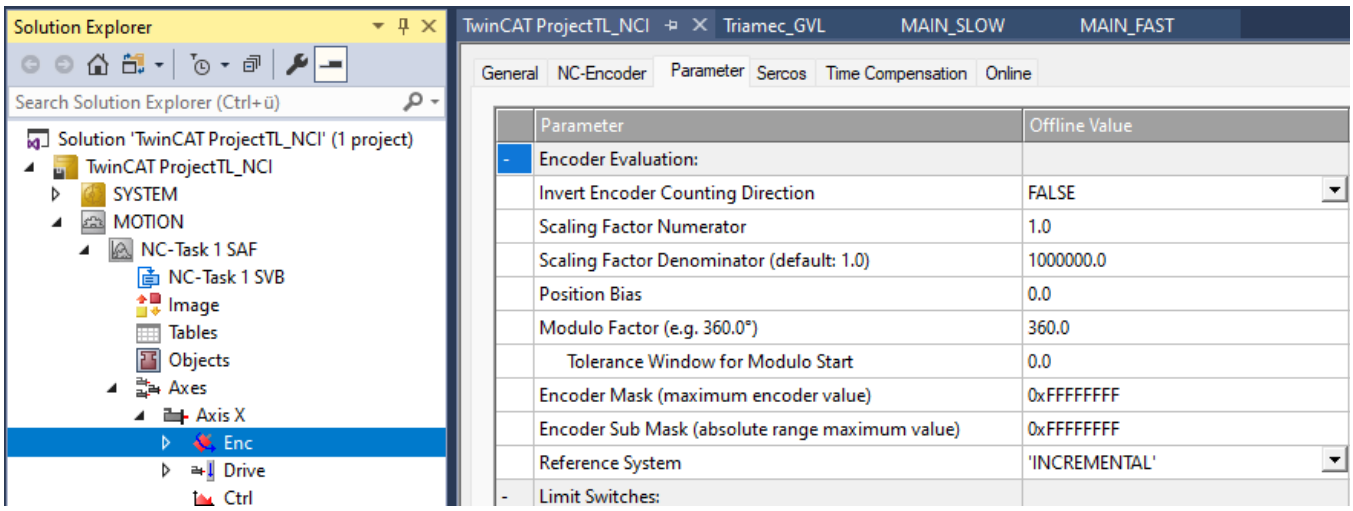


Figure 10: NCI encoder resolution parameter

Use the same factor with the unit conversion code, see also 2.5 and the following example.

To set a resolution of a linear axis to 1nm, apply the following procedure.

- **Axis N > Enc > Parameter > Scaling Factor Numerator = 1**.
- **Axis N > Enc > Parameter > Scaling Factor Denominator = 1000000**.
- Convert and scale the position values with `inc_per_unit = 1000000`:

```
(* get the coordinates from the NC/CNC and write it to the drive *)
fastPositionCmd := DINT_TO_LREAL(DWORD_TO_DINT(PosCmd)) / inc_per_unit;
SUPER^.CallFast( Trialink := Trialink );

(* get the actual position from the drive and pass it to NC *)
positionAct := ActualPositionFast(Trialink := Trialink);
(* coerse position to DINT range *)
IF ABS(positionAct * inc_per_unit) < 16#7FFFFFFF  THEN
  PosAct := DINT_TO_DWORD(LREAL_TO_DINT(positionAct * inc_per_unit));
ELSIF positionAct < 0 THEN
  PosAct := 16#80000000;
ELSE
  PosAct := 16#7FFFFFFF;
END_IF
```

The snippet above is an excerpt from the *NcTriaLink* example on *GitHub*, see chapter 5.
The interface to the *Triamec Lib* is at the instance calls to `TL_Axis2.ActualPositionFast()` and `TL_Axis2.fastPositionCmd`.

4 CNC Setup

For this chapter it's assumed that chapter 2 has been worked through.

4.1 Add SERCOS Devices

We use a dummy SERCOS Master and SERCOS Slaves to generate the correct CNC parameters for the axes. Later we will brake the connection and disable the devices.

1. Use the context menu **Add New Item...** of **I/O > Devices** to add a **SERCOS Master FC75xx**.
2. Use the context menu **Add New Item...** on the just added device to add **Drive Generic (SERCOS)** slave devices. Add as many as you have axes in your setup.
3. Change the Operation Mode of the drives in the **SERCOS Drive** tab to **Position 1 without Lag** (Figure 11).
4. Check that the following parameters are present and set correctly. Add missing parameters with **New…**

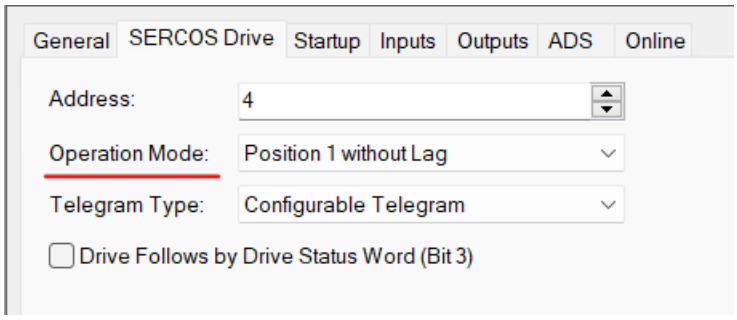


Figure 11: SERCOS Drive Operation Mode

- Tab **Startup:** S-0-0032, **Value** = 11
- Tab **Inputs**: S-0-0051
- Tab **Outputs**: S-0-0047
- Tab **Outputs**: S-0-0036

4.2 CNC Axes

To start add the **CNC Configuration** by selecting **Add new Item…** in the context menu of the **MOTION** node.

Then find the **Axes** node in the **CNC** module and add the amount of axes matching your setup, again through **Add new Item…** in the context menu.

At this point also the *Channels* can be added to the *CNC* module as for your purpose. Refer to the *Beckhoff* documentation for this topic. In compliance with the *Triamec* sample codes we add one named *Channel 1*.

Now go through the following instruction for each axis added before (see also Figure 12).

1. Link it to a drive added in the previous chapter at **Axis N > Configuration > Link To…**
   Select one of the **SERCOS Drives** showing up in the dialog.
2. Select a Channel at **Axis N > Configuration > Default Channel**.
3. Check the **Feed Axis** checkbox.

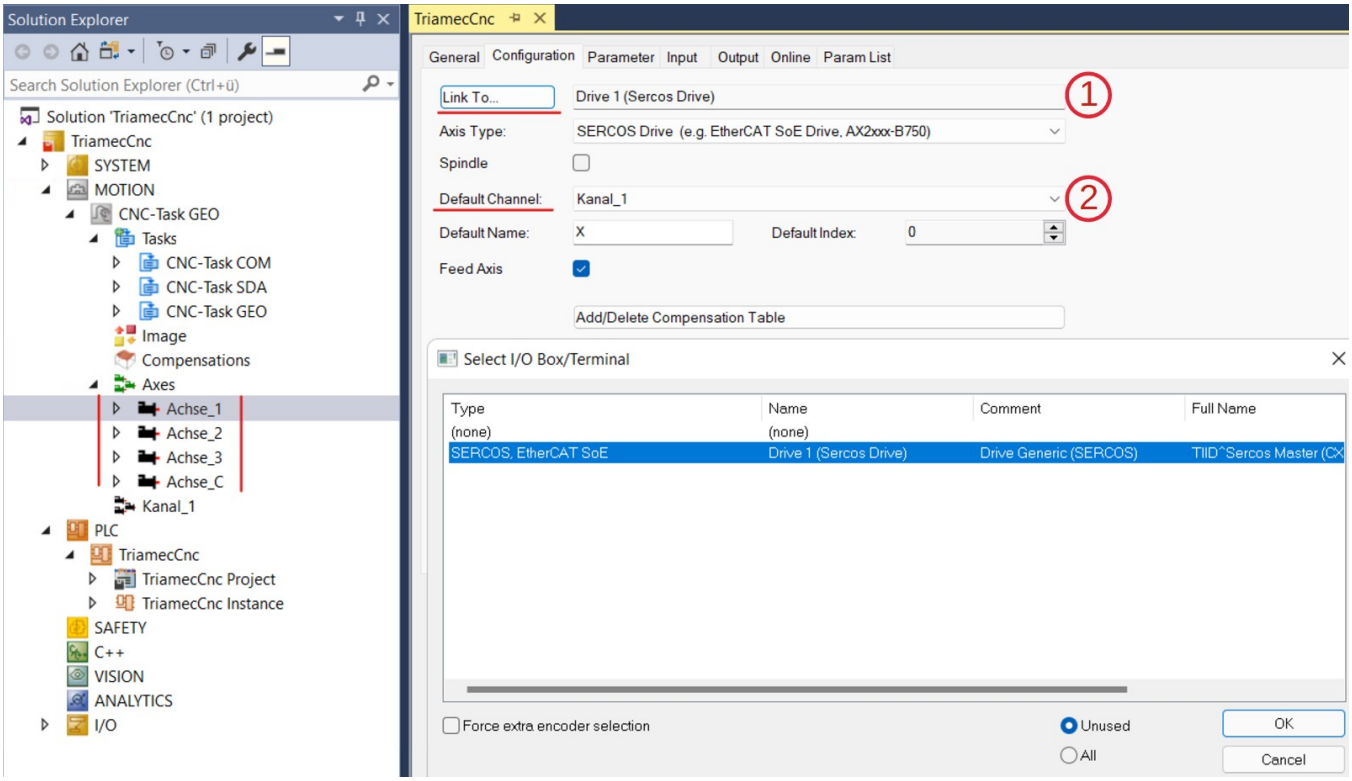


Figure 12: CNC axis configuration

Note The **Axis Type** should be configured automatically after linking. This also adds **Inputs** and **Outputs** to the CNC axes, which are needed later.

4.3 CNC Function Block

Here we introduce the function block *TL_CNC_Axis*.

It contains three sub-routines that implement the mapping from *CNC* to drive values and v.v.

The FB can be sourced from the *Release Package*. Add it with the context menu of the **Triamec** folder in the *PLC* project, by selecting **Add > Existing Item...**

Both *TL_CNC_Axis (FB)* make use of the *Beckhoff* library *Tc2_CncHli*. Add it to the *PLC* project through the context menu **Add library...** of the **References** folder within the project node.

4.4 CNC Tasks

The fast *PLC* Task must run at the same cycle time as the C*NC-Task GEO* because it receives the path planner data from the *CNC* and forwards it to the *Tria-Link* and vice versa.

To achieve this, right-click the **PLC > Project > TASK_FAST**, select **Assign to task** and choose the **CNC-Task GEO**. The result should look like in Figure 13.

This makes the *TASK_FAST* obsolete, which can be deleted under **SYSTEM > Tasks**.

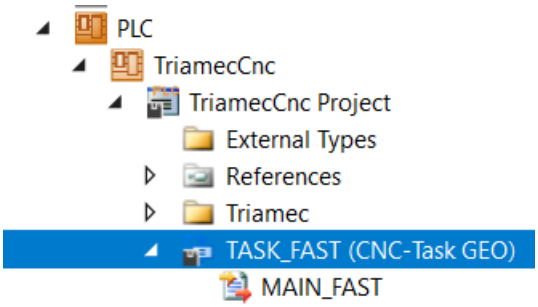


Figure 13: PLC Program MAIN_FAST assigned to GEO Task

A second method is described in chapter 8.5.

4.5 Linking Axes and PLC

Because the interface to the drive data is on *PLC* level, we need to make the following process variables to link to the *CNC* level. The process variables are also categorized into slow and fast data and must be used with the corresponding tasks. Ensure correct axis and index mapping.

4.5.1 SERCOS Interface (Slow Task)

Two process variables per axis are needed for the CNC to get control on the drive state.

```
SercosPhase  AT %Q* : WORD; (* link to: CNC > GEO > Axis > Inputs > Sercos Phase *)
SercosStatus AT %Q* : WORD; (* link to: CNC > GEO > Axis > Inputs > Drive status word *)
```

Basic state handling code is available in the Triamec examples, see chapter 5.

4.5.2 Position Data Exchange (Fast Task)

Two additional process variables per axis are made to exchange position data in the fast task.

```
SercosPosCmd AT %I* : DINT; (* link to: CNC > GEO > Axis > Outputs > Position command value *)
SercosPosAct AT %Q* : DINT; (* link to: CNC > GEO > Axis > Inputs > Position feedback 1 value *)
```

In the building process an instance of your *PLC* project is generated with the defined **Inputs** and **Outputs**. Use the context menu entry **Change Link …** on the variables to setup the links above. Check that no other links are active by clearing them via context menu entry **Clear Link(s)**.

Now the **I/O > Devices > Sercos Master** can be disabled through the context menu **Disable**.

4.6 Change Resolution

Adjust the following parameter at **MOTION > CNC > Axis_N > Parameter**.

```
getriebe[0].wegaufz       1        P-AXIS-00234 : Path resolution of the measuring system (num).
```

The unit of this parameter is 10'000 inc/mm or inc/degree. Its default value is 1.
Use the following equation for correct scaling in between *PLC* and *CNC*.

$$\text{inc\_per\_unit} = wegaufz * 10000$$

For example, to set a resolution of 1nm, we need 1'000'000 inc/mm. To achieve this we set `wegaufz = 100` and `inc_per_unit = 1000000` (see also chapter 2.5). Use the value to convert position data.

For examples see chapter 5 and the snippet in chapter 3.5.

5 Examples

To get started, *Triamec* provides examples for *CNC* and *NCI* setups on *GitHub*.

6 Writing PLC Code

The *PLC* sample code uses two objects of the *Triamec* library called in two *PRGs*.

- `TL_Trialink2` is the PCI adapter board object.
- `TL_Axis2` is the object that interacts with an axis.
- *MAIN* is the *PRG* for configuration and asynchronous state handling (Enable, Homing).
- *MAIN_FAST* is the *PRG* for fast data exchange in between *Tria-Link* and *TwinCAT* domains.

In standard mode, the path planner of *TwinCAT* sends positions to the drive. The following two function blocks of the *TriamecLib* allow using the path planner of the drive for axis movements. They are further described in [3].

- *TL_MC_MoveAbsolute*
- *TL_MC_MoveVelocity*

6.1 Axis Objects

All axes are defined and controlled by library axis objects of type `Triamec.TL_Axis2`. The axis object contains the following inputs.

- `enable` to enable the axis
- `stop` to force a stop as long as this input is `TRUE`
- `couple` to connect the axis with the *TwinCAT* path planner
- `referenceEnable` to enable homing
- `referenceStart` This is a trigger input to set `referenceEnable` to `TRUE`.
- `reset` to reset axis errors
- `iAxis` This logical axis number must be specified uniquely with a value between 1 and 32 and is usually the index of the array of `TL_Axis2` objects.

There is a simulation mode for axes. It is enabled by setting `gAxes[].Config.Simulate` to `TRUE` (see also chapter 2.4). In simulation mode, the state information is set independent of the real drive state which means the drive does not enable. However, the actual position information is correctly propagated if possible.

6.2 Path Planner and Coordinates

While the position controller always runs on the drive, the path planner can run either on the drive or in *TwinCAT*. After enabling, the axis starts up using the drive internal path planner.

Path Planner	Description
Drive	The path planner runs on the drive. Moving to a new position or starting a velocity move is commanded asynchronously by *PLCopen* function blocks. These are used i.e. for the homing routine.
TwinCAT	The NC/CNC or a custom PLC code runs the path planner. An axis enters *DirectCoupled* state if `TL_Axis2.couple` is set to `TRUE`. The axis then follows the setpoints commanded by *TwinCAT*.

See [3] for more details.

The following code block in *MAIN_FAST* is used for coordinate calculations and sharing position information between the *Tria-Link* and the *NC/CNC* in coupled mode.

```
FOR iAxis := 1 TO N_Axis DO
   gAxis[iAxis].fastPositionCmd := gCncAx[iAxis].CallFastPosCmd();
   gAxis[iAxis].CallFast( Trialink := Trialink );
   gCncAx[iAxis].CallFastPosAct(gAxis[iAxis].ActualPositionFast(Trialink:=Trialink));
END_FOR
```

Whereas the first two lines in the `FOR` loop send the commanded positions from the *NC/CNC* to the *Tria-Link*, and the 3rd line takes the actual position from the *Tria-Link* and sends it to the *NC/CNC*.

7 Register Access

With the *Tria-Link* library, we ship the possibility to reference register addresses by *URI* notation. Since the library version is decoupled from the drive firmware version, there are scenarios, where an *URI* is not available to a register of interest. Therefore it is also possible to directly reference a register by its *URI-Address* (hexadecimal number).

To do so from TwinCAT, the *URI-Address* can be found using the *TAM System Explorer*. Additionally the data type of the register must be known for correct transfer. The library ships global variables for the data type masks. Add the mask value to the *URI-Address* to mask it.

Datatype	Global Variable Name	Value
float 32bit	TL_REG_FLOAT_MASK	0x00400000
double 64bit	TL_REG_DOUBLE_MASK	0x00C00000

In reference to the example in 7.1.3, the register reference can look as following:

```
// reference the MotorTemperature by URI
readRegister.SetReg := gAxes[1].MC_Axis.register2.Axis.Signals.General.MotorTemperature;

// reference the MotorTemperature by URI-Address
readRegister.SetReg := 16#0022A000 + TL_REG_FLOAT_MASK
```

7.1 Acyclic

Two *Function Blocks* are available from the *TriamecLib* to read and write drive *Registers*. Both *Function Blocks* must be triggered with a positive edge on `Execute`.

Note Infrequent *Register* access with the following *Function Blocks* is limited to the slow *Task*. Calls from the fast *Task* result in undefined behavior.

7.1.1 TL_MC_RegisterWrite

Figure 14: FB for acyclic read of register values

Values must be written to an input of ***TL_MC_RegisterWrite***, depending on the *Register* data type.

- `SetBin` to write an integer (DWORD) or boolean (BOOL) value.
- `SetFloat` to write a float (LREAL) value.

Use the `setBin` input whenever `0 <> setReg AND TL_REG_FLOAT32_MASK`

Some parameters require activation (Committing). Write 1 to one or more of the following variables after writing these registers:

- General/Commands/CommitParameter
- Axes[]/Commands/Pathplanner/CommitParameter
- Axes[]/Commands/PositionController/CommitParameter
- Axes[]/Commands/CurrentController/CommitParameter

7.1.2 TL_MC_RegisterRead

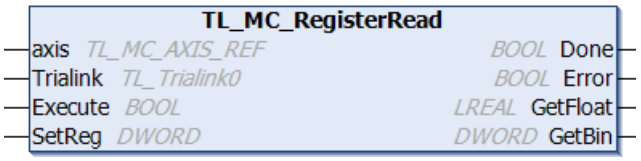


Figure 15: FB for acyclic polls of register values

The output of ***TL_MC_RegisterRead*** depends on the data type of the *Register* being read.

- `GetBin` to read an integer (DWORD) or boolean (BOOL) value.
- `GetFloat` to read a float (LREAL) or integer (DWORD) value.

7.1.3 Example

The sample code below sets the motor temperature limit and reads the motor temperature.

Declaration in *MAIN_SLOW*:

```
readRegister  : Triamec.TL_MC_RegisterRead; (* acyclic register access - read *)
writeRegister : Triamec.TL_MC_RegisterWrite; (* acyclic register access - write *)
```

Code snippet in *MAIN_SLOW*:

```
(* asynchronously read any drive register *)
readRegister.Execute := TRUE; // Toggle to read once
readRegister.SetReg  := gAxes[1].MC_Axis.register2.Axis.Signals.General.MotorTemperature;
readRegister(axis := gAxes[1].MC_axis, Trialink := gTrialink);

(* asynchronously write any drive register *)
writeRegister.Execute  := TRUE; // Toggle to write once
writeRegister.SetReg   := gAxes[1].MC_axis.register2.Axis.Parameters.Motor.TemperatureUpperLimit;
writeRegister.SetFloat := 80;
writeRegister(axis := gAxes[1].MC_axis, Trialink := gTrialink);
```

Note, that the `IN_OUT` variable named ***axis*** uses the *PLCopen* object ***.MC_axis*** of type ***TL_MC_AXIS_REF*** and not the general axis definition ***gAxes[n]*** of type ***TL_AxisSlow***.

Use the auto-complete feature in *TwinCAT* to find drive *Registers* and constants.

7.2 Committing Parameters

Changing *Registers* in a Commands tree takes immediate action. Whereas Registers in the Parameters tree require a commit to apply new values. This allows changing a set of *Parameters* and apply them all together.

There are different *Commit Groups*, where all have its dedicated commit command.

Commit Group	Command Register
General	General.Commands.CommitParameter
PathPlanner	Axes[].Commands.PathPlanner.CommitParameter
PositionController	Axes[].Commands.PositionController.CommitParameter
PathPlanner	Axes[].Commands.CurrentController.CommitParameter

To find the matching commit group for a Register, use the *TAM System Explorer*:

5. Find the target *Parameter* by navigating the *Register* tree and select it.
6. In the tab view, change to the **General** tab.
7. Find the value under **Identification > Tags > commitGroup**

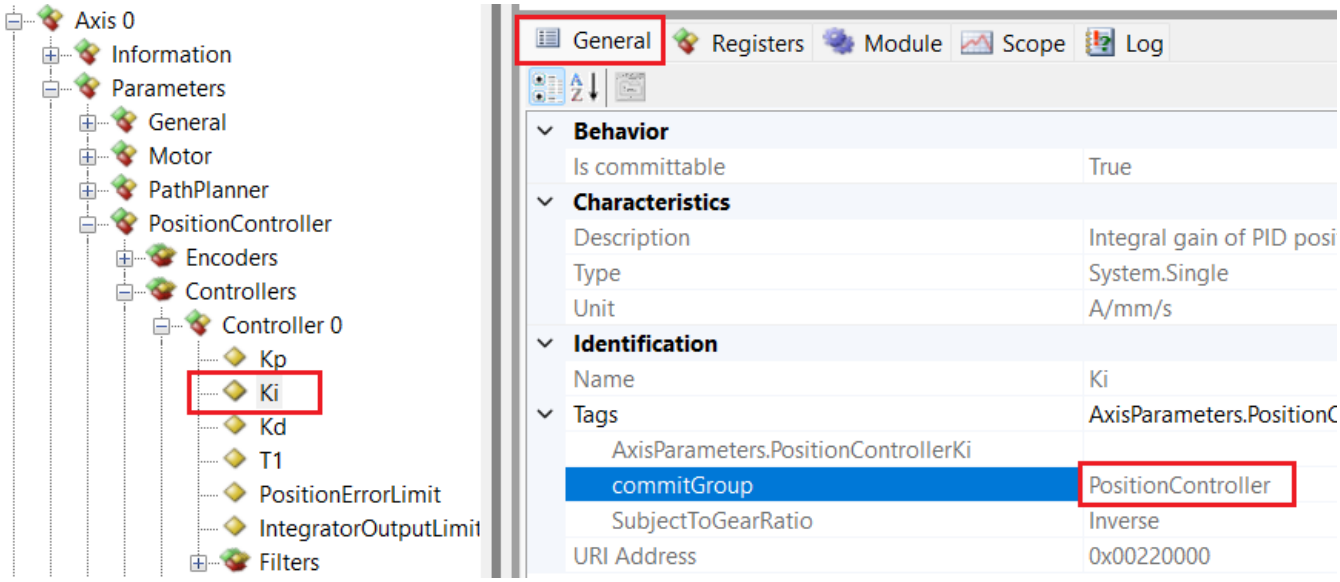


Figure 16: Find the Commit Group of a Parameter

All commit commands are Boolean, and reverted to False automatically, when the commit has finished.

Use the *TE_RegisterWriteInt32 FB* from the Library with the value 1, to set a commit command to true.

Warning	Do not change parameters before the last commit finished.
Warning	Write access of commitable parameters is not allowed for cyclic access.

7.3 Cyclic

Two *Function Blocks (FB)* are available from the *TriamecLib* to setup cyclic telegrams in between main

controller and drives. Both *FB* are controlled with the following two methods.

- `CallSlow()` Activate the configuration made with the *FB* inputs.
- `CallFast()` Exchange the configured values.

The following chapters describe each *FB*, with example code snippets in chapter 7.3.3.

7.3.1 TL_publishSlave2Master

Sets up a cyclic telegram, carrying up to four values, sourced from a drive register and sent to *TwinCAT*.

Note Values read with this method are in drive units. Especially with position related data, there is no correction with `axis.Config.GearFactor`.

Note When using *TL Adapter* cards without *DMA* (see 2.1), a restriction of max five `TL_publishSlave2Master` instances applies per TwinCAT Runtime. This restriction applies no matter if *DMA* is not supported or disabled by configuration (`Trialink.Config.DmaDisable = true`).

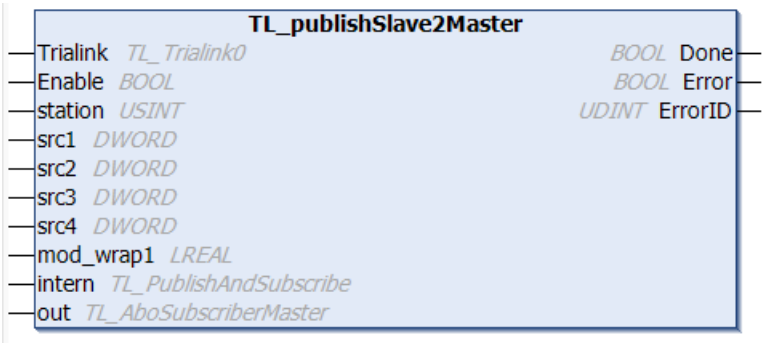


Figure 17: FB for cyclic slave to master telegrams

The four inputs `src1` to `src4` are configured with an *URI Address* of the register of interest. The values are then available in `out.val1` to `out.val4`. An example usage is shown in chapter 7.3.3, with the instance `publishS2M`.

Modulo

Exchanging register values which are subject to a modulo wrap need special consideration, because of integrated interpolation mechanisms. Examples are position setpoints and actual positions of a spindle.

Such values are only supported on the first channel of the telegram `src1`. The corresponding modulo range is set at the *FB* input `mod_wrap1`. The range must match the unit of the value exchanged. As this is usually position related data, define the modulo range with the drive registers in Axes[].Parameters.PathPlanner:

`mod_wrap1` = ModuloPositionMaximum – ModuloPositionMinimum

Interpolation

The default interpolation mode is linear, which uses the last two data points. Quadratic interpolation over three data points is available and can be set as follows. Use the index of the `src` input for the `interpolator` index.

Configure in slow *Task*:

```
publishS2M.out.interpolator[1].mode := InterpolPol2_XXX;
```

Time Shift

The data accessed in *TwinCAT* corresponds to the timestamp at the previous *Task* tick. The data may be shifted in time using the *FB* input setting `out.ShiftTicks`. The max recommended range is -1.0 to +1.0, which corresponds to a range of ± one fast *Task* tick.

Note Using positive values may result in extrapolation!

7.3.2 TL_publishMaster2Slave

Sets up a cyclic telegram, carrying up to five values, sent from *TwinCAT* to drive registers.

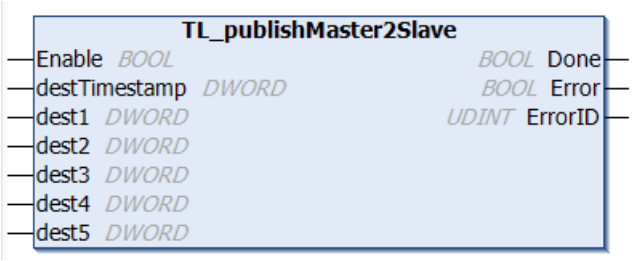


Figure 18: FB for cyclic master to slave telegrams

The five inputs `dest1` to `dest5` are configured with an *URI Address* of the register of interest. The values are then sent, by passing them to the `CallFast()` method. An example usage is shown in chapter 7.3.3, with the instance `publishM2S`.

7.3.3 Example

Declaration (e.g. in *Triamec_GVL*):

```
publishS2M      : Triamec.TL_publishSlave2Master; (* cyclic telegrams - publish slave to master *)
publishM2S      : Triamec.TL_publishMaster2Slave; (* cyclic telegrams - publish master to slave *)

myRealVar       : REAL; (* published value - master to slave *)
```

Configure in slow *Task*:

```
(* subscribe the actual current of an axis to the main controller *)
publishS2M.src1 := gAxes[1].MC_axis.register2.Axis.Signals.CurrentController.ActualCurrentQ;
publishS2M.station := gAxes[1].MC_axis.station;
publishS2M.CallSlow( Trialink:= gTrialink );

(* publish myRealVar to an application register in the drive *)
publishM2S.dest1 := gAxes[1].MC_axis.register2.Application.Variables.Floats;
publishM2S.CallSlow( Trialink:= gTrialink, gAxes[1] );
```

Cyclic call in the fast *Task* to write process variables:

```
(* cyclically read actual current into publishS2M.out.val1 *)
publishS2M.CallFast( Trialink:= Trialink );

(* cyclically update myRealVar, note that each publisher can transport up to 5 values *)
```

```
publishM2S.CallFast( TL_LREAL2DWORD(myRealVar), 0, 0, 0, 0, Trialink:= gTrialink );
```

8 Advanced Topics

8.1 Modulo

The modulo wrap value is set on the *TwinCAT* side and in the drive configuration. The values in the following example refer to a modulo wrap at 360 degrees.

Triamec Drive Parameters

Set the modulo range with the following drive registers.

- Axis[].Parameters.PathPlanner.ModuloPositionMaximum = 360
- Axis[].Parameters.PathPlanner.ModuloPositionMinimum = 0.0

TwinCAT PLC

In the *ConfigurationManager* function block, set the modulo travel.

```
TL_Axis2.Config.ModuloWrap := 360;
```

TwinCAT CNC

For details on *CNC* axis, spindle, and channel settings see the *Beckhoff CNC* documentation. If the axis is rotational with the *CNC* in units of 0.0001° and should get a 360° Modulo, set:

```
kenngr.achs_typ                 ROTATOR      (P-AXIS-00018 Type, 2 (ROTATOR) or 4 (SPINDLE))
kenngr.achs_mode                0x4          (P-AXIS-00015 Modulo)
getriebe[i].moduloo             3600000      (P-AXIS-00126 : [10-4degree] Upper modulo limit)
getriebe[i].modulou             0            (P-AXIS-00127 : [10-4degree] Lower modulo limit)
```

For CNC spindle mode consider Axis-Parameters (overwrite some settings above).

```
kenngr.achs_typ                 SPINDLE      (P-AXIS-00018 Type, 2 (ROTATOR) or 4 (SPINDLE))
kenngr.vb_prozent               100          (P-AXIS-00217 [0.1 %]Factor for speed reached)
getriebe[0].vb_min_null         1000         (P-AXIS-00216 [10-3Grad/s] Drehzahl null (Spindel))
getriebe[0].beschl_kennlinie.typ 0           (P-AXIS-00202 Type of the acceleration curve )
getriebe[i].moduloo             3600000      (P-AXIS-00126 : [10-4degree] Upper modulo limit)
antr.mode_act_pos               2            (P-AXIS-00122)
antr.mode_cmd_pos               2            (P-AXIS-00123)
antr.drive_encoder_range        3600000      ([1])
```

and channel parameters:

```
spdl_anzahl                     1            (P-CHAN-00082 Spindelanzahl)
main_spindle_ax_nr              1            (P-CHAN-00051 Logical axis number main spindle)
main_spindle_name               S            (P-CHAN-00053 Bezeichnung der Hauptspindel)
spindel[0].log_achs_nr          1            (P-CHAN-00036 Logical axis number of the Spindel)
spindel[0].bezeichnung          S1           (P-CHAN-00007 Bezeichnung der Spindel)
cax_face_id                     2            (Fräsmaschinen)
```

1 This value has to be scaled with getriebe[..].wegaufz and getriebe[..].wegaufn.

```
main_spindle_gear_change          0
```

TwinCAT NC

See *Beckhoff* documentation on using modulo moves.
Set the following values at **NC > Axis > Enc > Parameters**

- **Scaling Factor Numerator** = 1
- **Scaling Factor Denominator** = 10000 (same as `inc_per_unit` parameter, chapter 3.5)
- **Modulo Factor** = 360 (same as *Drive* setting)
- **Encoder Mask** = 0x36EE80 (*modulo ** `inc_per_unit` = 3600000)
- **Encoder Sub Mask** = 0xFFFFFF

Check also the *Beckhoff NCI* documentation on using modulo.

and in **NC > Axis > Drive > Parameter**:

- Set **Output Scaling Factor (Velocity)** = 2000
- Set **Axis > Settings > Unit** = **Degree**
- Activate the checkbox **Axis > Settings > Modulo**.

8.2 Referencing (Homing)

The axis module *TL_Axis2* contains a homing function which implements several homing procedures. The configuration is mostly dependent on the parameters with prefix `gAxis[N].Config.Reference`. Further details on homing and the implementation in *TwinCAT* can be found in [3] and [4].

8.3 TwinCAT Events

The *TwinCAT* interface supports the error concept for *Tria-Link* devices with release 3.9.0.0 or higher.

The axis object `TL_Axis2` contains the following state outputs for error feedback, updated with *TASK_SLOW*.

- `messageId` The drive error number
- `errorId` The library error number
- `sState` The library error as a string
- `error` `TRUE` if the library shows an `errorId`

The `errorId` shows library errors only and `messageId` shows drive errors or warnings.

Special care is taken to support existing applications. Two legacy modes are available. The mode *MessageForcedLegacy* is activated by setting:

```
TL_Axis2::Config.MessageForcedLegacy := TRUE;
```

The second mode *Automatic fallback* is active if the above switch is not set and one of the following conditions is met:

- DMA mode is off on this PCIe adapter (*TL*).

- A drive with firmware older than 4.15 is used.

The following table describes the outputs depending on the legacy modes.

	New mode	Automatic fallback	MessageForcedLegacy
errorId error	Library error codes only	Library and legacy drive error codes	
sState	Library error strings	Library and legacy drive error strings	
messageId	Drive message codes	0	

8.4 Error Display using TwinCAT Events

Errors are available at the function block output `ErrorId`. All errors are documented in [2].

If the error should be propagated using *TwinCAT* events, the error description file from the *Release Package*, `events/TriamecEventsTrialink*.xml`, must be installed as described in the `readme.txt` file in the same folder.

An `ErrorId` not equal to zero corresponds to an error if the `error` output of the *FB* is `TRUE`. Otherwise it is a warning or an information. Consider the sample codes on how to use events in *TwinCAT*.

8.5 CNC Task Setup

One special remark on the CNC task configuration to avoid real time problems:

If the *MAIN_FAST* program runs in a separate task, here called *TASK FAST*, *it* must run at the same cycle rate as the *CNC-Task GEO.*
If *TwinCAT* detects a real time issue called *Task Exceeds*, it may choose to skip the *TASK_FAST* once. In such a situation the *GEO* task might still run and generate the next point on the trajectory. Because *TwinCAT* does not run *TASK_FAST* this position value is not forwarded to the *Tria-Link*. This causes a jump in the trajectory. This problem can be avoided by ensuring that *GEO* and *TASK_FAST* are glued together. Use the following procedure to achieve this (Figure 19).

- Open the **Context** tab under **MOTION > CNC-Task GEO**.
- For **GEO 0** select **TASK_FAST** from the pull down menu in the **Task** column.
- With **Sort Order** the sequence of the calls can be configured. To call the **CNC-Task GEO** before the PLC sequence the **Sort Order** has to be set to a value between 0 and 100.

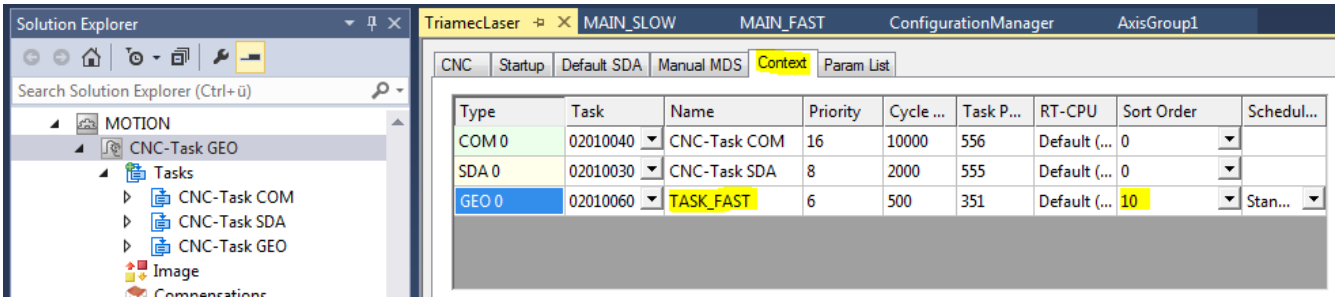


Figure 19: Context configuration of Task GEO

Glossary

GVL	Global Variable List, refer to Beckhoff TwinCAT documentation.
PLC	Programmable Logic Control, but mainly referring to the PLC node in TwinCAT projects.
GUI	Graphical User Interface

References

[1] “Servo Drive Setup Guide”, ServoDrive-SetupGuide_EP018.pdf, Triamec Motion AG, 2022.

[2] “TwinCAT Library: Tria-Link Messages”, AN103_TwinCAT-TriaLinkMessages_EP011.pdf, Triamec Motion AG, 2022.

[3] “Twincat Library: Using PLCopen”, AN108_TwinCAT-MotionControl_EP012.pdf, Triamec Motion AG, 2022.

[4] “Homing Procedures and Setup”, AN141_HomingProceduresAndSetup_EP003.pdf, Triamec Motion AG, 2022.

[5] “Triamec TwinCat Ethercat”, SWTC_TwinCAT-UserGuideEcat_EP009.pdf, Triamec Motion AG, 2022.

[6] Triamec Website, www.triamec.com, Triamec Motion AG, 2022.

[7] “FieldBus”, AN155_FieldBus_EP001.pdf, Triamec Motion AG, 2025

Revision History

Version	Date	Editor	Comment
022	2013-02-08	mvx	Add TLO100, move error codes to AN103. Use with sample code 3.0.
023	2014-03-28	mvx	New object gCnc for Task exceed compensation
024	2016-02-02	chm	Section 3.1: USB observer loop-back configuration
025	2019-09-23	dg	Synchronized with TSD Setup Guide EP001
026	2020-12-09	bl	Updated Nomenclature, removed references to TIOB
027	2021-06-23	mvx	Update and restructured for TwinCAT 3 and infos on how to attach GEO to MAIN_FAST
028	2023-01-26	sm	Update CD, restructure and integrate general setup guide
029	2023-02-14	sm	Remove spindle specific CNC Axis FB description (obsolete).
030	2023-04-24	sm, rb	Merge AN105 and AN109 Register Access into this guide, minor fixes
031	2023-09-26	sm, rb	Clean up static example references, interface descriptions at library level, rb ownership
032	2025-01-13	fm	Added reference to fieldbus doc AN155

Triamec Motion AG
Lindenstrasse 16
6340 Baar / Switzerland

Phone +41 41 747 4040
Email info@triamec.com
Web www.triamec.com

Disclaimer