



**TML\_LIB\_X20**

T E C H N O S O F T

**Motion Control Library for  
B&R X20**

## **User Manual**



# TECHNOSOFT

## **TML\_LIB\_X20 User Manual**

P091.040.X20.UM.0309

### **Technosoft S.A.**

Rue de Buchaux 38

CH-2022 BEVAIX

Switzerland

Tel.: +41 (0) 32 732 5500

Fax: +41 (0) 32 732 5504

[contact@technosoftmotion.com](mailto:contact@technosoftmotion.com)

[www.technosoftmotion.com/](http://www.technosoftmotion.com/)



---

## Read This First

Whilst Technosoft believes that the information and guidance given in this manual is correct, all parties must rely upon their own skill and judgment when making use of it. Technosoft does not assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.

All rights reserved. No part or parts of this document may be reproduced or transmitted in any form or by any means, electrical or mechanical including photocopying, recording or by any information-retrieval system without permission in writing from Technosoft S.A.

### ***About This Manual***

This book describes the motion library **TML\_LIB\_X20**. The TML\_LIB\_X20 is a **IEC61131-3 compatible** collection of motion function blocks, which can be integrated in a **B&R Automation Studio** application for the PLC family **B&R X20**. With TML\_LIB\_X20 motion library, you can quickly program the desired motion and control the Technosoft intelligent drives and motors (with the drive integrated in the motor case) from a B&R X20 PLC environment. The TML\_LIB\_X20 uses the **X20CS1070** communication module, through which the B&R PLC is connected with the Technosoft Drives/motors via a CAN bus link. The library supports both CAN-bus standards, the CAN 2.0A with 11-bit identifier and the CAN 2.0B with 29-bit identifier.

### ***Scope of This Manual***

This manual applies to the following Technosoft intelligent drives and motors:

- **IDM240 / IDM640** with firmware **F000H** or later (revision letter must be H or subsequent, i.e. I, J, etc.)
- **IDM680** (all models), with firmware **F500A** or later
- **ISCM4805 / ISCM8005** (all models), with firmware **F000H** or later
- **ISM4803** (CAN models), with firmware **F024H**
- **IBL2403** (CAN models), with firmware **F020H** or later
- **IPS110** (CAN models), with firmware **F005H** or later
- **IM23x** (models IS and MA), with firmware **F900H** or later

**IMPORTANT!** For a correct operation, these drives/motors must be programmed with firmware revision H or later, except for IDM680. **EasySetUp** Technosoft IDE for drives/motors setup,

---

<sup>1)</sup> **EasySetUp** is included in the **TML\_LIB\_X20** installation package, as a component of the **EasyMotion Studio Demo version**. EasySetup can also be downloaded free of charge from the Technosoft website.

---

*includes a firmware programmer with which you can check your drive/motor firmware version and revision and if needed, update your drive/motor firmware to revision H or later.*

The difference between the drives/motors with CAN 2.0B compliant protocol and those with the CAN 2.0A compliant protocol resides only in the firmware: all Technosoft products equipped with the CAN2.0A protocol have a firmware number starting with 2, i.e., the firmware code is **F2xxY** (where 2xx is the firmware number, and Y is the firmware revision).

### **Notational Conventions**

This document uses the following conventions:

- **Drive/motor** - an *intelligent drive* or an *intelligent motor* having the drive part integrated in the motor case
- **TML** – Technosoft Motion Language
- **IU** – drive/motor internal units
- **ACR.5** – bit 5 of ACR data
- **TMLCAN** – CAN communication protocol based on the CAN 2.0B standard. The TMLCAN is the default CAN-bus protocol for the Technosoft drives/motors without CANopen
- **TechnoCAN** – CAN communication protocol designed to permit the connection of Technosoft drives/motors without CANopen in a CANopen network, where messages are exchanged using CANopen protocol. TechnoCAN and CANopen do not hinder each other, and therefore can co-exist on the same physical bus.

### **Related Documentation**

**MotionChip™ II TML Programming** (part no. P091.055.MCII.TML.UM.xxxx) describes in detail TML basic concepts, motion programming, functional description of TML instructions for high level or low level motion programming, communication channels and protocols. Also give a detailed description of each TML instruction including syntax, binary code and examples.

**MotionChip II Configuration Setup** (part no. P091.055.MCII.STP.UM.xxxx) describes the MotionChip II operation and how to setup its registers and parameters starting from the user application data. This is a technical reference manual for all the MotionChip II registers, parameters and variables.

**Help of the EasySetUp software** – describes how to use **EasySetUp** to quickly setup any Technosoft drive for your application using only 2 dialogs. The output of EasySetUp is a set of setup data that can be downloaded into the drive EEPROM or saved on a PC file. At power-on, the drive is initialized with the setup data read from its EEPROM. With EasySetUp it is also possible to retrieve the complete setup information from a drive previously programmed. EasySetUp includes a firmware programmer with allows you to update your drive firmware to the latest revision. **EasySetUp can be downloaded free of charge from Technosoft web page**

---

**Help of the EasyMotion Studio software** – describes how to use the EasyMotion Studio to create motion programs using in Technosoft Motion Language (TML). EasyMotion Studio platform includes **EasySetUp** for the drive/motor setup, and a **Motion Wizard** for the motion programming. The Motion Wizard provides a simple, graphical way of creating motion programs and automatically generates all the TML instructions. *With EasyMotion Studio you can fully benefit from a key advantage of Technosoft drives – their capability to execute complex motions without requiring an external motion controller, thanks to their built-in motion controller. A demo version of EasyMotion Studio (with EasySetUp part fully functional) can be downloaded free of charge from Technosoft web page.*

***If you Need Assistance ...***

---

<b>If you want to ...</b>	<b>Contact Technosoft at ...</b>
Visit Technosoft online	World Wide Web: <a href="http://www.technosoftmotion.com/">http://www.technosoftmotion.com/</a>
Receive general information or assistance	World Wide Web: <a href="http://www.technosoftmotion.com/">http://www.technosoftmotion.com/</a> Email: <a href="mailto:contact@technosoftmotion.com">contact@technosoftmotion.com</a>
Ask questions about product operation or report suspected problems	Fax: (41) 32 732 55 04 Email: <a href="mailto:hotline@technosoftmotion.com">hotline@technosoftmotion.com</a>
Make suggestions about or report errors in documentation	

---

---

*This page is empty*



---

# Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
<b>2</b>	<b>Getting started.....</b>	<b>3</b>
2.1	Hardware installation .....	3
2.2	Software installation.....	4
2.2.1	Installing EasySetUp.....	4
2.2.2	Installing TML_LIB_X20 library.....	4
2.3	Drive/motor setup.....	4
2.4	Build a B&R Automation Studio project with TML_LIB_X20 .....	5
<b>3</b>	<b>TML_LIB_X20 description .....</b>	<b>11</b>
3.1	Basic concepts.....	11
3.2	TML_LIB_X20 requirements.....	13
3.3	Internal units and scaling factors .....	14
3.4	TML data.....	14
3.5	Axis ID Identification .....	14
3.6	Functions descriptions .....	15
3.6.1	FB TS_MoveAbsolute.....	17
3.6.2	FB TS_MoveRelative.....	19
3.6.3	FB TS_MoveAdditive .....	21
3.6.4	FB TS_MoveSCurveAbsolute.....	22
3.6.5	FB TS_MoveSCurveRelative.....	24
3.6.6	FB TS_MoveVelocity .....	26
3.6.7	FB TS_SetPVT .....	27
3.6.8	FB TS_PVTPoint .....	30
3.6.9	FB TS_SetPT.....	32
3.6.10	FB TS_PTPoint.....	35
3.6.11	FB TS_Homing .....	37
3.6.12	FB TS_Stop .....	39
3.6.13	FB TS_ExternalAnalogue .....	40
3.6.14	FB TS_ExternalDigital .....	42
3.6.15	FB TS_ExternalOnLine.....	43
3.6.16	FB TS_GearIn.....	45

---

3.6.17	FB MC_GearOut.....	47
3.6.18	FB MC_CamTableSelect.....	48
3.6.19	FB TS_CamIn .....	49
3.6.20	FB MC_CamOut .....	52
3.6.21	FB TS_SetMaster .....	53
3.6.22	FB TS_Power .....	55
3.6.23	FB TS_Reset .....	56
3.6.24	FB TS_ResetDrive.....	57
3.6.25	FB TS_WriteLongParameter .....	58
3.6.26	FB TS_WriteFixedParameter .....	59
3.6.27	FB TS_WriteIntegerParameter .....	60
3.6.28	FB TS_ReadActualPosition .....	61
3.6.29	FB TS_ReadLongParameter .....	62
3.6.30	FB TS_ReadFixedParameter .....	63
3.6.31	FB TS_ReadIntegerParameter .....	64
3.6.32	FC TS_BuildMessage.....	65
3.6.33	FC TS_InterpretMessage .....	66
3.6.34	FC TS_InitAxis.....	67
3.6.35	FC TS_SendCommand .....	68

---

# 1 Introduction

The programming of Technosoft intelligent drives/motors involves 2 steps:

- 1) Drive/motor setup
- 2) Motion programming

For **Step 1 – drive/motor setup**, Technosoft provides **EasySetUp**. EasySetUp is an integrated development environment for the setup of Technosoft drives/motors. The output of EasySetUp is a set of *setup data*, which can be downloaded to the drive/motor non-volatile memory or saved on your PC for later use. The setup data is copied at power-on into the RAM memory of the drive/motor and is used during runtime. The reciprocal is also possible i.e. to retrieve the complete setup data from a drive/motor non-volatile memory previously programmed. EasySetUp can be downloaded free of charge from Technosoft web page. It is also provided on the TML\_LIB\_X20 installation CD.

For **Step 2 – motion programming**, Technosoft offers multiple options, like:

- 1) Use the drives/motors embedded motion controller and do the motion programming in Technosoft Motion Language (TML). For this operation, Technosoft provides **EasyMotion Studio**, an IDE for both drive setup and motion programming. The output of EasyMotion Studio is a set of setup data and a TML program to download and execute on the drive/motor.
- 2) Use a **.DLL** with high-level motion functions which can be integrated in a host application written in C/C++, Delphi Pascal, Visual Basic or LabVIEW
- 3) Use a **IEC61131-3 compatible** library with motion function blocks which can be integrated in a PLC application based on one of the IEC 61131-3 standard languages
- 4) Combine Option 1) with Options 2) or 3) to really distribute the intelligence between the master/host and the drives/motors in complex multi-axis applications. Thus, instead of trying to command each step of an axis movement, you can program the drives/motors using TML to execute complex tasks and inform the master when these tasks have been done.

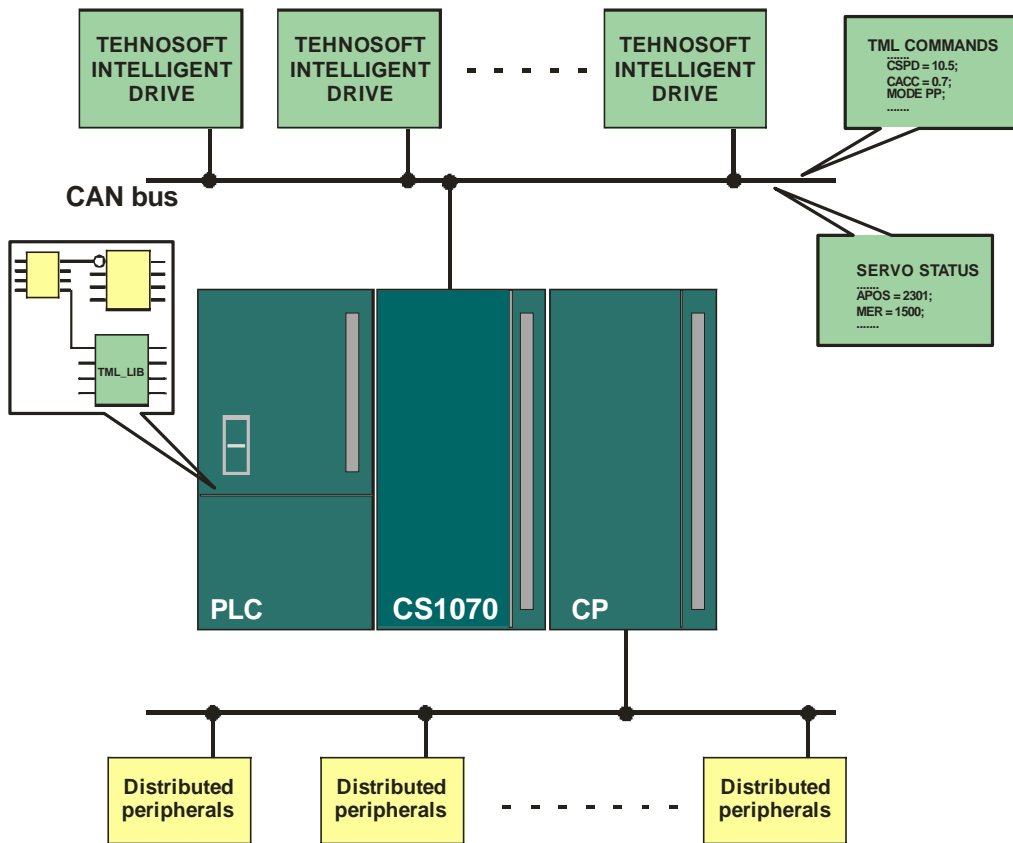
**TML\_LIB\_X20** is part of Option 3). It is an IEC61131-3 compatible motion library developed for the **B&R X20** family of PLCs. The link between Technosoft drives/motors and the PLC is done via the CAN-bus, using as interface the **X20CS1070** communication module. This solution embeds a Technosoft motion control unit in the X20 PLC environment, providing simultaneous access to precise motion control and a large number of distributed peripherals (Figure 1.1).

This manual shows how to install and use the components of the **TML\_LIB\_X20** library. These were built following the guidelines described in PLC standards for motion control.

The functionality and the interface of these additional functions comply with the technical specifications of the standard.

## **Remarks:**

- *Option 4) requires using **EasyMotion Studio** instead of EasySetUp. With EasyMotion Studio you can create high-level motion functions in TML, to be called from your PLC*
- *EasyMotion Studio is also recommended if your application includes a homing, as it comes with 32 predefined homing procedures to select from and adapt if necessary.*

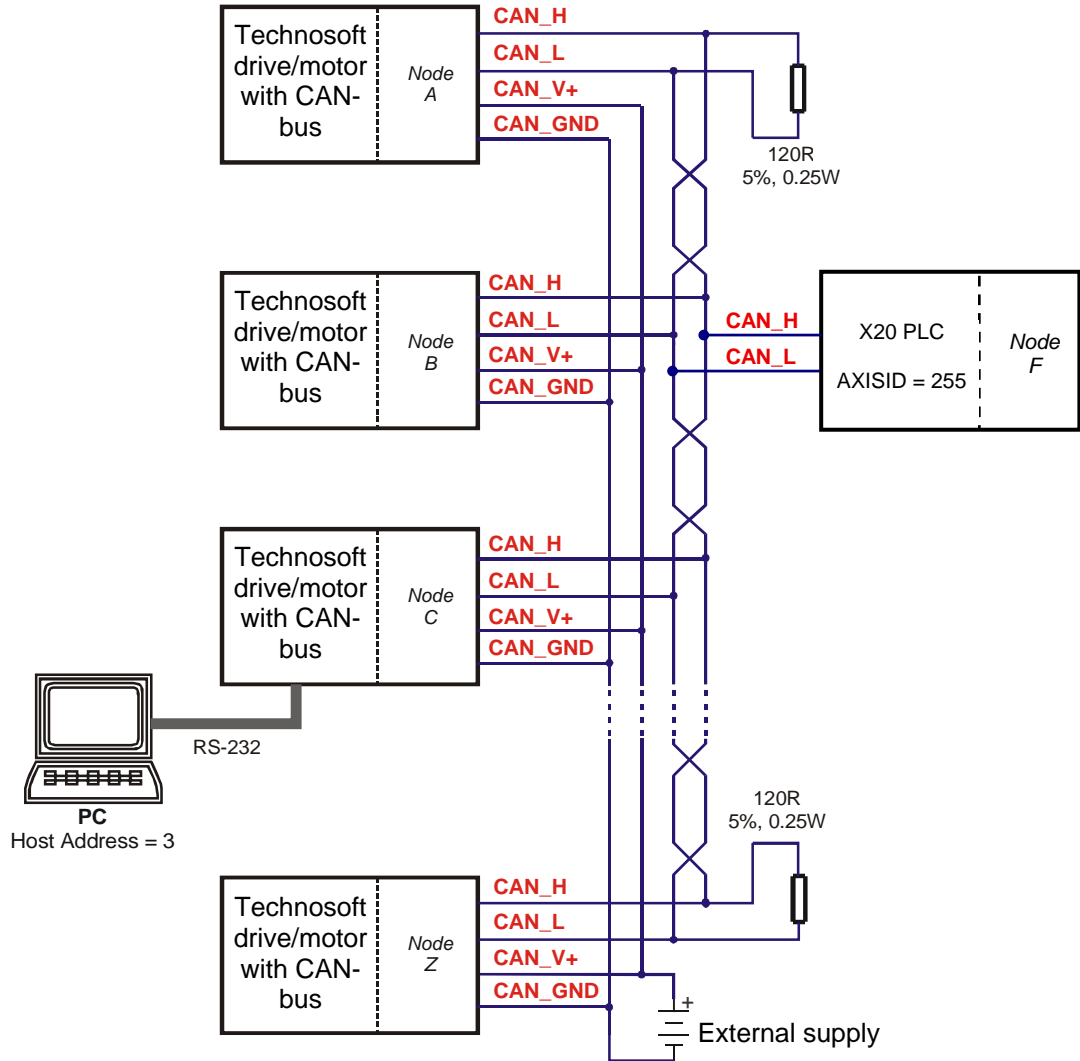


**Figure 1.1.** Tehnosoft drives/motors connections with an X20 B&R PLC

## 2 Getting started

### 2.1 Hardware installation

Figure 2.1 shows how to connect via a CAN-bus link several drives/motors with the B&R X20 PLC + the X20CS1070 communication module.



**Figure 2.1** Multiple-Axis CAN network

**Remarks:**

- The 120Ω terminating resistors between CAN-HI and CAN-LO are mandatory
- Some Technosoft products do not require an external supply for the CAN interface

- 
- *In comparison with TMLCAN, TechnoCAN has the following restrictions:*
    - *The maximum number of axes is 31: possible Axis ID values: 1 to 31*
    - *The maximum number of groups is 5: possible Group ID values: 1 to 5*

For the hardware installation of the Technosoft drives/motors see their user manual.

For the hardware installation of the X20CS1070 communication module see the user manual provided by B&R together with the product.

For drives/motors setup, you can connect your PC to any drive/motor using an RS232 serial link. Through this serial link you can access all the drives/motors from the network. Alternately, you can connect your PC directly on the CAN bus network if it is equipped with a CAN interface supported by EasySetUp. See the on-line help of EasySetUp for a list with CAN interfaces supported.

## 2.2 Software installation

In order to perform successfully the following software installations, make sure that you have the "Administrator" rights.

### 2.2.1 Installing EasySetUp

On the TML\_LIB\_X20 installation CD you'll find the setup for EasyMotion Studio Demo version. This application includes a fully functional version of EasySetUp and a demo version of EasyMotion Studio. Start the setup and follow the installation instructions.

### 2.2.2 Installing TML\_LIB\_X20 library

Start the TML\_LIB\_X20 setup and follow the installation instructions.

## 2.3 Drive/motor setup

Before starting to send motion commands from your PLC, you need to do the drive/motor setup according with your application needs. For this operation you'll use **EasySetUp**.

**EasySetUp** is an integrated development environment for the setup of Technosoft drives and motors (with the drive integrated in the motor case). The output of **EasySetUp** is a set of setup data, which can be downloaded to the drive/motor non-volatile memory or saved on your PC for later use.

A setup contains all the information needed to configure and parameterize a Technosoft drive/motor. This information is preserved in the drive/motor non-volatile memory in the setup table. The setup table is copied at power-on into the RAM memory of the drive/motor and is used during runtime. The reciprocal is also possible i.e. to retrieve the complete setup data from a drive/motor non-volatile memory previously programmed

### Steps to follow for commissioning a Technosoft drive/motor

#### Step 1. Start EasySetUp

From Windows Start menu execute: "Start | Programs | EasySetUp | EasySetUp" or "Start | Programs | EasyMotion Studio | EasySetUp" depending on which installation package you have used.

---

## Step 2. Establish communication

EasySetUp starts with an empty window from where you can create a **New** setup, **Open** a previously created setup which was saved on your PC, or **Upload** the setup from the drive/motor.

Before selection one of the above options, you need to establish the communication with the drive/motor you want to commission. Use menu command **Communication | Setup** to check/change your PC communication settings. Press the **Help** button of the dialogue opened. Here you can find detailed information about how to setup your drive/motor and do the connections. Power on the drive/motor and then close the **Communication | Setup** dialogue with OK. If the communication is established, EasySetUp displays in the status bar (the bottom line) the text "**Online**" plus the axis ID of your drive/motor and its firmware version. Otherwise the text displayed is "**Offline**" and a communication error message tells you the error type. In this case, return to the Communication | Setup dialogue, press the Help button and check troubleshoots.

**Remark:** When first started, EasySetUp tries to communicate with your drive/motor via RS-232 and COM1 (default communication settings). If your drive/motor is powered and connected to your PC port COM1 via an RS-232 cable, the communication can be automatically established.

## Step 3. Setup drive/motor

Press **New** button and select your drive/motor type. Depending on the product chosen, the selection may continue with the motor technology (for example: brushless motor, brushed motor) or the control mode (for example stepper – open-loop or stepper – closed-loop) and type of feedback device (for example: incremental encoder, SSI encoder)

This opens 2 setup dialogues: for **Motor Setup** and for **Drive setup** through which you can configure and parameterize a Technosoft drive/motor, plus several predefined control panels customized for the product selected.

In the **Motor setup** dialogue you can introduce the data of your motor and the associated sensors. Data introduction is accompanied by a series of tests having as goal to check the connections to the drive and/or to determine or validate a part of the motor and sensors parameters. In the **Drive setup** dialogue you can configure and parameterize the drive for your application. In each dialogue you will find a **Guideline Assistant**, which will guide you through the whole process of introducing and/or checking your data. Close the Drive setup dialogue with **OK** to keep all the changes regarding the motor and the drive setup.

## Step 4. Download setup data to drive/motor

Press the **Download to Drive/Motor** button to download your setup data in the drive/motor non-volatile memory in the *setup table*. From now on, at each power-on, the setup data is copied into the drive/motor RAM memory that is used during runtime. It is also possible to **Save** the setup data on your PC and use it in other applications.

## Step 5. Reset the drive/motor to activate the setup data

## 2.4 Build a B&R Automation Studio project with TML\_LIB\_X20

The TML\_LIB\_X20 library is designed to work with B&R Automation Studio v3.0 SP3.

The library is provided as a collection of structured text source files. For each library component, function or function block, you will find a source code file with extension **.st** identifiable by the file

---

name. The next steps detail how to include the TML\_LIB\_X20 components in an Automation Studio project.

1. **Create a B&R Automation Studio project.** Launch **B&R Automation Studio** and create a new project. For details on how to create the project read the online help of B&R Automation Studio.
2. **Configure X20CS1070 communication module.** Select Physical View of the project, open the **I/O Configuration** window of X20CS1070 module and make the changes listed below, the remaining settings must remain with the default values.
  - a. Set **Function model** to default (flat).
  - b. Set the **CAN bus baud-rate**. The value selected must match the baud rate set on Technosoft drives/motors (the default value for Technosoft drives/motors is 500kbaud).
  - c. Set the **CAN data object format** to UINT type.
3. **Add TML\_lib data types.** TML\_lib library uses the following structures to store various information across the function blocks:
  - a. **Axis\_TS** – structure containing the data and status of a Technosoft axis at PLC level
  - b. **CANbuffer** – structure containing the pointers to X20CS1070 registers
  - c. **PTref** – structure containing the data for a PT point
  - d. **PVTref** – structure containing the data for a PVT point

**Remarks:**

- *The Axis\_TS and CANbuffer types are mandatory. The PTref and PVTref are required only when the PLC application programs the drive/motor for PT or PVT mode.*
- *The data types should be declared as global data type within the Global.typ file.*

To insert the types open the \*.fun file as text and then copy the data type declarations from TML\_lib.typ and insert them in the file.

4. **Add global variables required by TML\_lib to access to X20CS1070 registers.** TML\_lib it's using pointers of global variables to access X20CS1070 registers (transmission and reception). The user must define the following:
  - a. RxCANData\_TS and TxCANData\_TS variables - **Array [0..4]** of type **UDINT** used to access the CAN Rx and Tx data
  - b. RxCANIdentifiert\_TS and TxCANIdentifier\_TS variables of type **UDINT** used to access the CAN Rx and Tx identifier
  - c. RxCANParam\_TS and TxCANParam\_TS variables - **Array [0..1]** of type **USINT** used to access the control registers of CAN Rx and CAN Tx buffers.
  - d. RxCountLatch and TxCountLatch variables - **Array [0..1]** of type **USINT** used to access the latch registers of CAN Rx and CAN Tx buffers.



Name	Type	Constant	Value	Description [1]
* COPYRIGHT - Technosoft				
* File: Global.var				
* Author: g_blujdea				
* Created: February 21, 2009				
* Global variables of project TML_lib				
RxCANIdentifier_TS	UDINT	<input type="checkbox"/>	0	
TxCANIdentifier_TS	UDINT	<input type="checkbox"/>	0	
RxCANData_TS	UINT[0..4]	<input type="checkbox"/>	0	
TxCANData_TS	UINT[0..4]	<input type="checkbox"/>	0	
RxCANParam_TS	USINT[0..1]	<input type="checkbox"/>	0	
TxCANParam_TS	USINT[0..1]	<input type="checkbox"/>	0	
RxCountLatch	USINT[0..1]	<input type="checkbox"/>	0	
TxCountLatch	USINT[0..1]	<input type="checkbox"/>	0	
RxBuffer	CANBuffer	<input type="checkbox"/>	0	
TxBuffer	CANBuffer	<input type="checkbox"/>	0	

**Figure 2.** Global variables required by TML\_lib\_X20 to read/write X20CS1070 registers

**5. Map the global variables over the X20CS1070 module registers:**

+ RXCount	USINT	Automatic	RxCountLatch[0]
+ RXCountLatch	USINT	Automatic	RxCountLatch[1]
+ TXCountReadBack	USINT	Automatic	TxCountLatch[0]
+ TXCountLatchRead...	USINT	Automatic	TxCountLatch[1]
+ RXDataSize0	USINT	Automatic	RxCANParam_TS[0]
+ RXIdent0	UDINT	Automatic	RxCANIdentifier_TS
+ RXData0Word0	UINT	Automatic	RxCANData_TS[0]
+ RXData0Word1	UINT	Automatic	RxCANData_TS[1]
+ RXData0Word2	UINT	Automatic	RxCANData_TS[2]
+ RXData0Word3	UINT	Automatic	RxCANData_TS[3]
→ TXCount	USINT	Automatic	TxCANParam_TS[0]
→ TXDataSize	USINT	Automatic	TxCANParam_TS[1]
→ TXIdent	UDINT	Automatic	TxCANIdentifier_TS
→ TXDataWord0	UINT	Automatic	TxCANData_TS[0]
→ TXDataWord1	UINT	Automatic	TxCANData_TS[1]
→ TXDataWord2	UINT	Automatic	TxCANData_TS[2]
→ TXDataWord3	UINT	Automatic	TxCANData_TS[3]

**Figure 3.** Global variables mapping

- 
6. **Add global variable to store Technosoft axes information.** Axes information must be stored in an array of type **Axis\_TS**. The number of elements in the array must be equal with the number of Technosoft axes.
  7. **Init the CANbuffer structures with the pointers of global variables.** The pointers of the global variables used to access the X20CS1070 must be grouped in 2 global variables of type **CANbuffer**, one for reception and one for transmission. Also, each axis requires the pointer to the transmission buffer. For correct operation of the library the initialization must be done before calling any TML\_lib\_X20 function or function block.

```

PROGRAM _INIT

    RxBuffer.pData := ADR(RxCANData_TS[0]);
    RxBuffer.pIdentifier := ADR(RxCANIdentifier_TS);
    RxBuffer.pParameter := ADR(RxCANParam_TS);
    RxBuffer.pLatchRegister := ADR(RxCountLatch);

    TxBuffer.pData := ADR(TxCANData_TS[0]);
    TxBuffer.pIdentifier := ADR(TxCANIdentifier_TS);
    TxBuffer.pParameter := ADR(TxCANParam_TS);
    TxBuffer.pLatchRegister := ADR(TxCountLatch);

    TS_Axes[0].CANInterface := ADR(TxBuffer);
    TS_Axes[0].AxisID := 255;
    TS_Axes[1].CANInterface := ADR(TxBuffer);
    TS_Axes[1].AxisID := 123;
    TS_Axes[2].CANInterface := ADR(TxBuffer);
    TS_Axes[2].AxisID := 222;
    TS_Axes[3].CANInterface := ADR(TxBuffer);
    TS_Axes[3].AxisID := 25;

END_PROGRAM

```

*Figure 4. CANbuffer initialization*

8. **Add TML\_LIB\_X20 source files.** Select **Insert | Add Object...** menu command to start the Add Object wizard and add the function and function blocks required by your application. From the Add Object wizard, select **File** category and then **Existing File** from templates. Press Next button and browse for the source files installation folder (default C:\Program Files\Technosoft\TML\_LIB\_X20\Library). Once reached the installation folder select the functions required by your application and press Open button.

**Remarks:**

- *The TS\_BuildMessage and TS\_InterpretMessage functions are mandatory for correct operation of the library so they must be included in all projects.*
- *The TS\_BuildMessage and TS\_InterpretMessage functions have 2 source files each, for each CAN-bus protocol, TechnoCAN or TMLCAN. The name of the source files are suffixed with the CAN identifier length for easier identification of the protocol supported.*

- 
9. **Functions and function blocks declaration.** Each function or function block must be declared in the \*.fun file associated to the project. Open the \*.fun file as text and then copy the function block declarations from TML\_lib.fun and insert them in the file.

---

*This page is empty*

---

## 3 TML\_LIB\_X20 description

### 3.1 Basic concepts

The TML\_LIB\_X20 library allows you to implement motion control applications in a B&R Automation Studio project with minimal knowledge of Technosoft Motion Language. With the functions and the function blocks you can control the Technosoft drive/motors individually or grouped by sending commands for simple (TS\_MoveAbsolute), complex (TS\_SetPVT, TS\_PVTPoint) or synchronized (TS\_GearIn) motions. The library is completed by several non-motion driving functions and functions blocks necessary for drive/motor administration (TS\_Power, TS\_WriteIntegerParameter).

The TML\_LIB\_X20 elements were built following the guidelines described in PLC standards for motion control and developed using IEC61131-3 programming languages.

Each Technosoft drive/motor is represented at the PLC level using the term **Axis**. Its behavior is defined through up to 8 states. During operation an axis can be in only one state. Figure 3.1 presents the axis behavior implemented in TML\_LIB\_X20 and the function blocks which have influence on the states. The current state for each axis is stored on the PLC, in an **Axis\_TS** structure. Several axes must be grouped in an **array** of type **Axis\_TS**. Table 3.1 shows the axis states and their corresponding values.

*Table 3.1 Axis states*

State	State	State byte value
Disabled	0	1
StandStill	1	2
Discrete Motion	2	4
Continuous Motion	3	8
Synchronized Motion	4	16
Stopping	5	32
ErrorStop	6	64
Homing	7	128

When the drive/motor is powered up its state is set to **Disabled**. In this state the drive/motor power stage is disabled and none of motion function blocks is executed. During **Disabled** state you must call **TS\_InitAxis** and **TS\_Power**. The **TS\_InitAxis** function enables the drive/motor to send automatically its status information towards the PLC. This feature is required for proper operation of TML\_LIB\_X20 components. With **TS\_Power** function block the power stage of the drive/motor is enabled and, if no error occurs, the axis is transferred to **StandStill** state. If the drive/motor responds with an error message, the axis is transferred to **ErrorStop** state. When the axis reaches the state **StandStill** you can start sending motion commands. From this moment each motion function block transfers the axis in its associated state. Administrative functions (no driving motion) have no influence on axis state.

The axis is transferred to **ErrorStop** if on the drive/motor an error occurs due to:

- Invalid setup data
- Protection triggered
- Control error

The motion function block cannot be executed when the axis is ErrorStop state. From this state the axis can be transferred to **Standstill** state with function block **TS\_Reset** or to **Disabled** state with function block **TS\_ResetDrive**. Function block **TS\_Reset** resets all drive/motor internal error and it must be followed by a **TS\_Power** call to re-enables the power stage. With function block **TS\_ResetDrive** you perform a reset of the drive/motor. If the drive/motor parameters were changed during normal operation and you want to use the same parameters after drive/motor reset, execute function **TS\_SaveParameters** prior issuing **TS\_ResetDrive** call. The function **TS\_SaveParameters** saves the drive/motor actual parameters in the non-volatile memory and thus enabling you to use them after a drive/motor reset or power off.

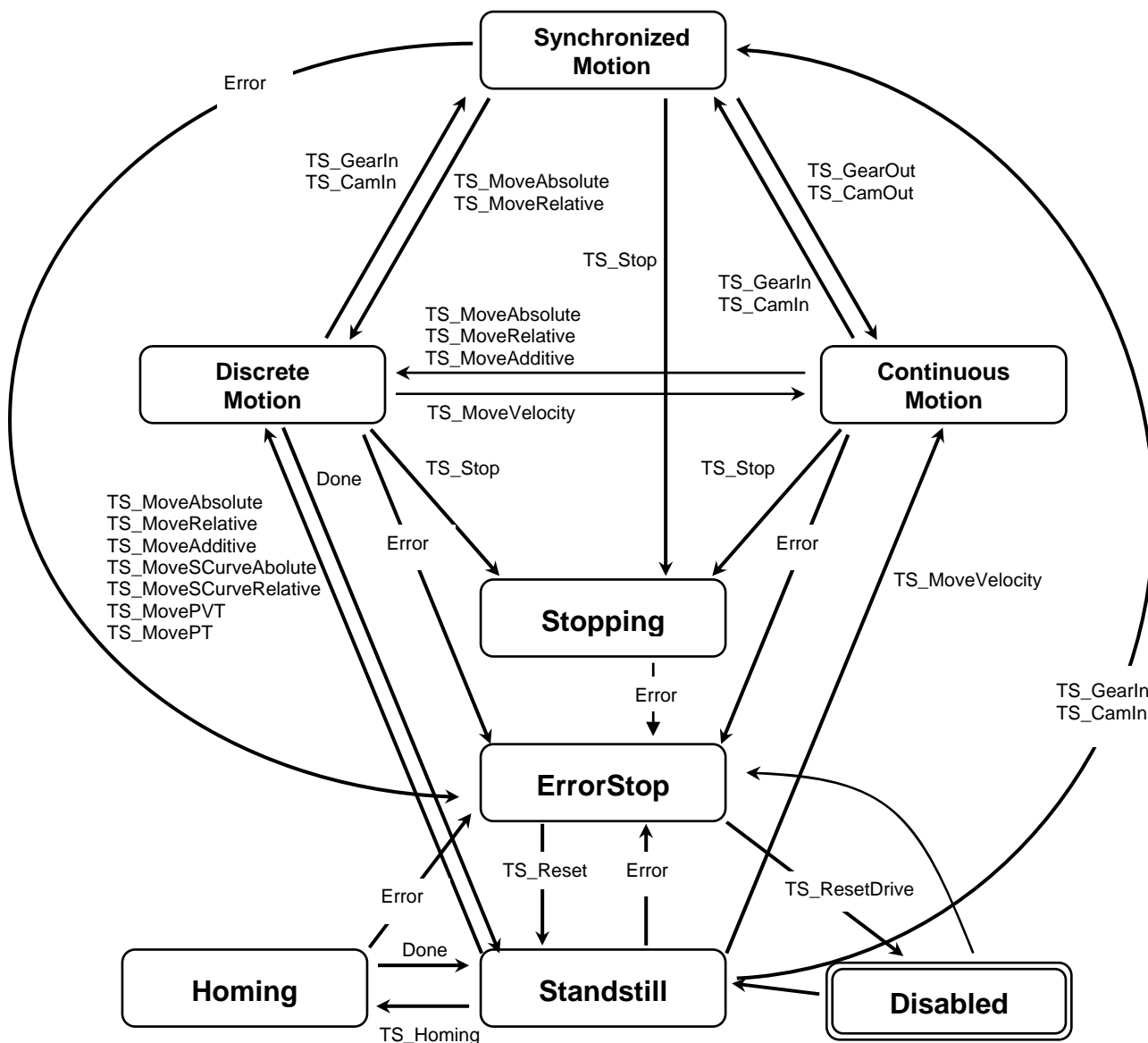


Figure 3.1 Axis behavior

The TML commands issued by the functions and function blocks are encapsulated according with the CAN protocol selected with TS\_BuildMessage function. When the TS\_BuildMessage function is called it updates the registers of the CAN module and triggers the transmission of the CAN message.

The CAN messages received by the X20CS1070 module are stored in the reception registers and from there are read by TS\_InterpretMessage function which analyzes the message and saves the data in the array with axes information. The information stored in the array is available to all library components.

**Remark:** The user must map the CAN module registers to PV. TO ADD MORE DETAILS

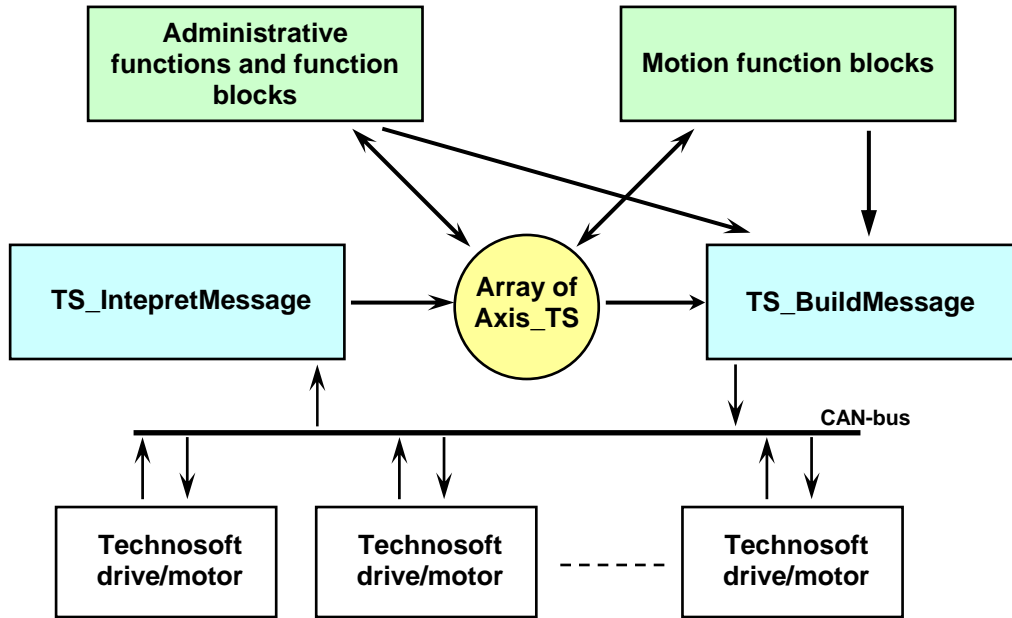


Figure 3.2. TML\_LIB\_X20 functionality

### 3.2 TML\_LIB\_X20 requirements

In order to use TML\_LIB\_X20 for B&R PLC you must have the following minimal hardware configuration:

- One or more Technosoft drives/motors, connected through a CAN-bus network
- A B&R X20 PLC
- A B&R X20CS1070 CAN module

The software required in order to implement an application on the previous configuration consists of:

- **EasySetUp** for setup of Technosoft drives/motors
- Automation Studio V3.0 + SP3 for hardware configuration and programming of the PLC

---

### 3.3 Internal units and scaling factors

Technosoft drives/motors work with parameters and variables represented in internal units (IU). The parameters and variables may represent various signals: position, speed, current, voltage, etc. Each type of signal has its own internal representation in IU and a specific scaling factor. In order to easily identify each type of IU, these have been named after the associated signals. For example the **position units** are the internal units for position, the **speed units** are the internal units for speed, etc.

The scaling factor of each internal unit shows the correspondence with the international standard units (SI). The scaling factors are dependent on the product, motor and sensor type. Put in other words, the scaling factors depend on the setup configuration.

In order to find the internal units and the scaling factors for a specific case, select the application in the project window and then execute menu command **Help | Help Topics | Setup Data Management | Internal Units and Scaling Factors**.

**Important:** *The help topic provides customized information, function of the application setup. If you change the drive, the motor technology or the feedback device, check again the scaling factors with this command. It may show you other relations!*

### 3.4 TML data

The TML uses the following data types:

- `int`            16-bit signed integer
- `uint`           16-bit unsigned integer
- `fixed`          32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part.
- `long`            32-bit signed integer
- `ulong`           32-bit unsigned integer

Since the motion defining parameters, i.e. position (distance), speed, acceleration and jerk have different data types in TML, for consistency, they are represented at the PLC level using only data type **real**, the conversion to TML data types is performed automatically by the function block.

### 3.5 Axis ID Identification

The data exchanged on the CAN bus is done using messages. Each message contains one TML instruction to be executed by the receiver of the message. Apart from the binary code of the TML instruction attached, any message includes information about its destination: an axis (drive/motor) or a group of axes. This information is grouped in the **Axis/Group ID Code**. Each drive/motor has its own 8-bit Axis ID and Group ID.

**Remarks:**

1. *The Axis ID of a drive/motor must be **unique** and is set during the drive/motor setup phase with **EasySetup**. The `TML_LIB_X20` doesn't contain a function or function block for changing the value of Axis ID.*
2. *A drive/motor belongs, by default, to the group ID = 1.*



- 
3. The Axis ID and Group ID of a drive/motor are stored in TML variable **AAR**(uint@0x030C). Use **TS\_ReadIntegerParameter** to read the value of the Axis ID and Group ID.

The Group ID represents a way to identify a group of axes, for a multicast transmission. This feature allows the PLC to send a command simultaneously to several axes, for example to start or stop the axes motion in the same time. When a function block sends a command to a group, all the axes members of this group will receive the command. For example, if the axis is member of group 1 and group 3, it will receive all the messages that in the group ID include group 1 and group 3.

Each axis can be programmed to be member of one or several of the 8 possible groups.

**Table 3.2** Definition of the groups

Group No.	Group ID value
1	1 ( 0000 0001b )
2	2 ( 0000 0010b )
3	4 ( 0000 0100b )
4	8 ( 0000 1000b )
5	16 ( 0001 0000b )
6	32 ( 0010 0000b )
7	64 ( 0100 0000b )
8	128 (1000 0000b )

**Remark:** In comparison with TMLCAN, TechnoCAN has the following restrictions:

- The maximum number of axes is 31: possible Axis ID values: 1 to 31
- The maximum number of groups is 5: possible Group ID values: 1 to 5

By default the function blocks send the commands to one axis. Several function blocks can also send commands to a group of axes. The destination of the command (single axis or group of axes) is set through function block's input **ControlWord**. If ControlWord.15 is set when the function block is executed the commands are sent to the axes having the Group ID equal with the value read from input **Axis**.

**Remark:** If your application requires group commands, add an *Axis\_TS* structure for a "dummy" axis in the array with axes information. In the first word of the structure set the Group ID you want to use.

## 3.6 Functions descriptions

The section presents the functions and function blocks implemented in the **TML\_LIB\_X20** library.

The implemented function blocks are classified as:

- Motion related
  - TS\_MoveAbsolute
  - TS\_MoveRelative
  - TS\_MoveAdditive
  - TS\_MoveSCurveAbsolute
  - TS\_MoveSCurveRelative
  - TS\_MoveVelocity
  - TS\_ExternalAnalogue
  - TS\_ExternalDigital

- 
- TS\_ExternalOnline
  - TS\_SetPVT
  - TS\_PVTPoint
  - TS\_SetPT
  - TS\_PTPoint
  - TS\_GearIn
  - TS\_GearOut
  - TS\_CamIn
  - TS\_CamOut
  - TS\_Stop
  - TS\_Homing
  - Administrative (no driving motion).
    - TS\_ReadActualPosition
    - TS\_ReadStatus
    - TS\_Power
    - TS\_CamTableSelect
    - TS\_SetMaster
    - TS\_WriteLongParameter
    - TS\_WriteFixedParameter
    - TS\_WriteIntegerParameter
    - TS\_ReadLongParameter
    - TS\_ReadFixedParameter
    - TS\_ReadIntegerParameter
    - TS\_ResetFault
    - TS\_ResetDrive
    - TS\_AxisStatus
    - TS\_SendCommand
    - TS\_SaveParameters

For each function you will find the following information:

- The structured text prototype
- Parameters description with their name and associated data type
- A functional description
- Remarks regarding drive/motor setup
- An example that illustrates how to use the function or function block

**Remark:** The library components type, i.e. function and function blocks, is specified in the description title. A function block title starts with **FB**, while for a function it starts with **FC**.

---

### 3.6.1 FB TS\_MoveAbsolute

#### ST prototype:

TS\_MoveAbsolute(BOOL Execute, REAL Position, REAL Velocity, REAL Acceleration, Axis\_TS AxisDescription);

#### Parameters description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	Position	REAL	Position to reach expressed in TML position internal units
	Velocity	REAL	The slew speed expressed in TML speed internal units
	Acceleration	REAL	Acceleration increment expressed in TML acceleration internal units.
Input/Output	AxisDescription	STRUCT	Axis information
Output	Done	BOOL	Commanded position reached
	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	Signal the function block is waiting for motion complete
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs an absolute positioning with trapezoidal speed profile. You specify the position to reach plus the velocity (maximum travel) and the acceleration/deceleration rate. The velocity and acceleration must be positive. Negative values are taken in modulus.

On detecting a rising edge at the **Execute** input, the function block starts sending motion commands and sets the **Busy** output. The **Busy** output remains set until the drive/motor signals target reached, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor. If a drive's error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

During motion execution the axis is transferred into **DiscreteMotion** state, when the target is reached (**Done** output is set) the axis is transferred to **Standstill** state.

#### Remarks:

1. *The function block requires drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*

- 
2. *If the application requires switching between discrete motion (position control) and continuous motion (speed control) you must also close the speed loop and perform the tuning of the speed controller.*

---

### 3.6.2 FB TS\_MoveRelative

#### ST prototype:

TS\_MoveRelative(BOOL Execute, REAL Distance, REAL Velocity, REAL Acceleration, Axis\_TS AxisDescription);

#### Parameters description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	Distance	REAL	Position increment expressed in TML position internal units
	Velocity	REAL	The slew speed expressed in TML speed internal units
	Acceleration	REAL	Acceleration rate expressed in TML acceleration internal units.
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Signal the motion complete
	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	Signal the function block is waiting for motion complete
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs a relative positioning with trapezoidal speed profile. You specify the position increment plus the velocity (maximum travel speed) and the acceleration/deceleration rate. The values of velocity and acceleration must be positive. Negative values are taken in modulus.

On detecting a rising edge at the **Execute** input, the function block starts sending motion commands and sets the **Busy** output. The **Busy** output remains set until the drive/motor signals target reached, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor. If a drive's error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

During motion execution the axis is transferred to **DiscreteMotion** state, when the target is reached (**Done** output is set) the axis is transferred to **Standstill** state.

#### Remarks:

1. *The function block requires drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*

- 
2. *If the application requires switching between discrete motion (position control) and continuous motion (speed control) you must also close the speed loop and perform the tuning of the speed controller.*

---

### 3.6.3 FB TS\_MoveAdditive

#### ST prototype:

TS\_MoveAdditive(BOOL Execute, REAL Distance, REAL Speed, REAL Acceleration, Axis\_TS AxisDescription);

#### Parameters description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	Distance	REAL	Position increment expressed in TML position internal units
	Velocity	REAL	The slew speed expressed in TML speed internal units
	Acceleration	REAL	Acceleration rate expressed in TML acceleration internal units.
Input/Output	AxisDescription	Axis_TS	Axis information
	Done	BOOL	Signal the motion complete
Output	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	Signal the function block is waiting for motion complete
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs a relative positioning with trapezoidal speed profile. The position to reach is computed by adding the position increment to the previous position to reach, independently of the moment when the command was issued. You specify the position increment plus the velocity (maximum travel speed) and the acceleration/deceleration rate. The values of velocity and acceleration must be positive. Negative values are taken in modulus on the drive/motor.

On detecting a rising edge at the **Execute** input, the function block starts sending motion commands and sets the **Busy** output. The **Busy** output remains set until the drive/motor signals target reached, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor. If a drive's error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

During motion execution the axis is transferred to **DiscreteMotion** state, when the target is reached (**Done** output is set) the axis is transferred to **Standstill** state.

---

### 3.6.4 FB TS\_MoveSCurveAbsolute

#### ST prototype:

TS\_MoveSCurveAbsolute(BOOL Execute, REAL Position, REAL Speed, REAL Acceleration, REAL Jerk, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	Position	REAL	Position to reach expressed in TML position internal units
	Velocity	REAL	The slew speed expressed in TML speed internal units
	Acceleration	REAL	Acceleration rate expressed in TML acceleration internal units.
	Jerk	REAL	Acceleration maximum jerk expressed in TML jerk internal units
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Signal the motion complete
	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	Signal the function block is waiting for motion complete
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs an absolute positioning with an S-curve shape of the speed. This shape is due to the jerk limitation, leading to a trapezoidal or triangular profile for the acceleration and an S-curve profile for the speed. You specify the position to reach plus the velocity (maximum travel speed), the maximum acceleration/deceleration rate and the jerk rate.

The function block can be executed only from **Standstill** state. During motion the parameters cannot be changed. Therefore when executing successive motions with TS\_MoveSCurveAbsolute function blocks, you should wait for the previous motion to end before setting the new motion parameters and starting next motion.

When the motion is stopped with function block TS\_Stop, the deceleration phase is done smooth, using an S-curve speed profile.

On detecting a rising edge at the **Execute** input, the function block starts sending motion commands and sets the **Busy** output. The **Busy** output remains set until the drive/motor signals target reached, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor. If a drive's error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.



---

During motion execution the axis is transferred to **DiscreteMotion** state, when the target position is reached (**Done** output is set) the axis is transferred to **Standstill** state.

**Remarks:**

1. *The function block requires drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*
2. *If the application requires switching between discrete motion (position control) and continuous motion (speed control) you must also close the speed loop and perform the tuning of the speed controller.*

---

### 3.6.5 FB TS\_MoveSCurveRelative

#### ST prototype:

TS\_MoveSCurveRelative(BOOL Execute, REAL Distance, REAL Speed, REAL Acceleration, REAL Jerk, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	Distance	REAL	Position increment expressed in TML position internal units
	Speed	REAL	The slow speed expressed in TML speed internal units
	Acceleration	REAL	Acceleration rate expressed in TML acceleration internal units.
	Jerk	REAL	Acceleration's maximum jerk expressed in TML jerk internal units
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	The motion is complete
	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	Signal the function block is waiting for motion complete
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs a relative positioning with an S-curve shape of the speed. This shape is due to the jerk limitation, leading to a trapezoidal or triangular profile for the acceleration and an S-curve profile for the speed. You specify the position to reach plus the velocity (maximum travel speed), the maximum acceleration/deceleration rate and the jerk rate.

The function block can be executed only from **Standstill** state. During motion the parameters should not be changed. Therefore when executing successive motions with TS\_MoveSCurveAbsolute function blocks, you should wait for the previous motion to end before setting the new motion parameters and starting next motion.

When the motion is stopped with function block TS\_Stop, the deceleration phase is done smooth, using an S-curve speed profile.

On detecting a rising edge at the **Execute** input, the function block starts sending motion commands and sets the **Busy** output. The **Busy** output remains set until the drive/motor signals target reached, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor. If a drive's error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

---

During motion execution the axis is transferred to **DiscreteMotion** state, when the target position is reached (**Done** output is set) the axis is transferred to **Standstill** state.

**Remarks:**

1. *The function block requires the drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*
2. *If the application requires switching between discrete motion (position control) and continuous motion (speed control) you must also close the speed loop and perform the tuning of the speed controller.*

---

### 3.6.6 FB TS\_MoveVelocity

#### ST prototype:

TS\_MoveVelocity(BOOL Execute, REAL Velocity, REAL Acceleration, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	Velocity	REAL	The jog speed expressed in TML speed internal units
	Acceleration	REAL	Acceleration rate expressed in TML acceleration internal units.
Input/Output	AxisDescription	Axis_TS	Axis information
Output	InVelocity	BOOL	Signal that commanded velocity reached
	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	The function block is waiting for velocity to reach the target speed command
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block commands a trapezoidal speed profile.

At the rising edge of the **Execute** input, the motion command is sent and the output **Busy** is set. The axis is transferred to ContinuousMotion state. When the jog speed is reached, the **Done** output is set and **Busy** is reset.

The **CommandAborted** is set if another function block starts sending motion commands. If a drive's motion error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**. All outputs are reset when **Execute** is reset.

#### Remarks:

1. *The function block requires drive/motor speed loop to be closed. During the drive/motor setup, in the **Drive Setup** dialogue, select **Speed** at Control Mode and perform the speed controller tuning.*
2. *If the application requires switching between continuous motion (speed control) and discrete motion (position control) the position loop must be closed, also. In the **Drive setup** dialogue select **Position** at **Control Mode**, then enter the **Advanced** dialogue and close the speed loop. After the selection perform the tuning of the position controller and speed controller.*

---

### 3.6.7 FB TS\_SetPVT

#### ST prototype:

TS\_SetPVT(BOOL Execute, REAL PVTPOS0, INT LowLevelValue, INT PVTCounterValue, UINT ControlWord, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	At rising edge the command is sent
	PVTPOS0	REAL	The initial position for absolute PVT mode
	LowLevelValue	INT	The new value for low buffer signaling
	PVTCounterValue	INT	The new counter value for the first PVT point
	ControlWord	WORD	Selects the block options
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Signal the path described through PVT points ended correctly, the last point has velocity zero
	CommandAborted	BOOL	The motion command is aborted by another motion command
	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function programs a drive/motor to work in PVT motion mode. In PVT motion mode the drive/motor performs a positioning path described through a series of points. Each point specifies the desired **P**osition, **V**elocity and **T**ime, i.e. contains a PVT data. Between the points the built-in reference generator performs a 3rd order interpolation.

**Remark:** The function block just programs the drive/motor for PVT mode, the motion mode is activated from function block TS\_PVTPoint, and therefore the TS\_PVTPoint must follow TS\_SetPVT call. Also, the function block TS\_PVTPoint sends the PVT points to the drive/motor.

A key factor for getting a correct positioning path in PVT mode is to set correctly the distance in time between the points. Typically this is 10-20ms, the shorter the better. If the distance in time between the PVT points is too big, the 3rd order interpolation may lead to important variations compared with the desired path.

The PVT motion mode can be started only when the previous motion is complete. However, you can switch at any moment to another motion mode.

The PVT mode can be relative (ControlWord.0 = 0) or absolute (ControlWord.0 = 1). In the absolute mode, each PVT point specifies the position to reach. The initial position may be either the current position reference TML variable **TPOS** (ControlWord.12=1) or a preset value read from the TML parameter **PVTPOS0** (ControlWord.12=0). In the relative mode, each PVT point specifies the position increment relative to the previous point. In both cases, the time is relative to the previous point i.e. represents the duration of a PVT segment. For the first PVT point, the time is measured from the starting of the PVT mode.

Each time when the drive receives a new PVT point, it is saved into the PVT buffer. The reference generator empties the buffer as the PVT points are executed. The PVT buffer is of type FIFO (first in, first out). The default length of the PVT buffer is 7 PVT points. Each entry in the buffer is made up of 9 words, so the default length of the PVT buffer in terms of how much memory space is reserved is 63 (3Fh) words. The drive/motor automatically sends messages to the PLC when the buffer is full, low or empty. The messages contain the PVT status (TML variable **PVTSTS**). The buffer full condition occurs when the number of PVT points in the buffer is equal with the buffer size. The buffer low condition occurs when the number of PVT points in the buffer is less or equal with a programmable value. When ControlWord.7=1 the buffer low level is programmed with the value read from **LowLevelValue** input. The buffer empty condition occurs when the buffer is empty and the execution of the last PVT point is over. When the PVT buffer becomes empty the drive/motor:

- Remains in PVT mode if the velocity of last PVT point executed is zero and waits for new points to receive
- Enters in quick stop mode if the velocity of last PVT point executed is not zero

Therefore, a correct PVT sequence must always end with a last PVT point having velocity zero.

**Remarks:**

1. The PVT and PT modes share the same buffer. Therefore the TML parameters and variables associated with the buffer management are the same.
2. Both the PVT buffer size and its start address are programmable via TML parameters (int@0x0864) and PVTBUFLen (int@0x0865). Therefore if needed, the PVT buffer size can be substantially increased. Use TS\_WriteIntegerParameter to change the PVT buffer parameters.

**Table 3.3.** ControWord bits description

Bit	Value	Description
0	0	The PVT mode is relative
	1	The PVT mode is absolute
1	0	Target update mode 1. Generates new trajectory starting from the actual values of position and speed reference (i.e. don't update the reference values with load/motor position and speed)
	1	Target update mode 0. Generates new trajectory starting from the actual values of load/motor position and speed (i.e. update the reference values with load/motor position and speed)
6-2	0	Reserved
7	0	No change in the buffer low parameter
	1	Change the buffer low parameter with the value specified in <b>LowLevelValue</b>
11-8	0	Reserved
12	0	If PVT mode is set as absolute, the initial position is taken from TML parameter PVTPOS0 (default = 0). The initial position is used to compute the distance to move up to the first PVT point.
	1	If PVT mode is set as absolute, the initial position is taken from TML variable <b>TPOS</b> . The initial position is used to compute the distance to move up to the first PVT point.
13	0	No change to the internal integrity counter
	1	Change internal integrity counter with the value specified in <b>PVTCounterValue</b>
14	0	Integrity checking is active
	1	Integrity checking inactive
15	0	Nothing

On detecting a rising edge at **Execute** input the function block reads the inputs and sends the motion commands to the drive/motor. The output **Busy** is set and remains set until the function block **TS\_PVTPoint** signals "motion mode active" moment when the **Done** output is set and **Busy** reset.

**Remarks:**

1. *The function block requires the drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*
2. *If the application requires switching between discrete motion (position control) and continuous motion (speed control) the speed loop must be closed, also. In the **Drive setup** dialogue select **Position** at **Control Mode**, then enter the **Advanced** dialogue and close the speed loop. After the selection perform the tuning of the position controller and speed controller.*

---

### 3.6.8 FB TS\_PVTPoint

#### ST prototype:

TS\_PVTPoint(BOOL Execute, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, PVTRef PVT\_REF, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	At rising edge the command is sent
	PVT_REF	PVTRef	Description of PVT point
	AxisDescription	Axis_TS	Axis information
Input/Output	Done	BOOL	Signal that commanded velocity is reached
	CommandAborted	BOOL	The motion command is aborted by another motion command
Output	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error message was received from the drive/motor while <b>!Done</b>
	ErrorID	WORD	Information about the error occurred
	PVTStatus	WORD	The PVT status send by the drive/motor

**Description:** The function block activates the PVT motion mode and sends the PVT points to a drive/motor previously programmed with function block TS\_SetPVT to work in PVT motion mode. In PVT motion mode the drive/motor performs a positioning path described through a series of points. Each point specifies the desired **Position**, **Velocity** and **Time**, i.e. contains a PVT data. Between the points the built-in reference generator performs a 3rd order interpolation.

The PVTRef structure contains the following elements:

Component	Data type	Description
Position	REAL.	Represents the position to be reached at the end of the PVT segment
Velocity.	REAL	Is the velocity at the of the PVT segment
Time	INT	Represents the time interval of the PVT segment. The maximum time interval is 511 IU.
Counter	INT	The counter value for the current point. The maximum value for the counter is 127.

On detecting a rising edge at **Execute** input the function block reads the inputs and sends the PVT point to the drive/motor. The output **Busy** is set and remains set until the drive signals the motion is complete, moment when the **Done** output is set and **Busy** reset. If the last point sent to has non-zero velocity the drive/motor signals the condition quick stop and the function block sets the output **Error**.

The function block sends a PVT point every time detects a rising edge at input **Execute**.

When the PLC receives a message from the drive/motor containing the PVT status, the function transfers its value to output **PVTStatus**.



**Table 3.4** PVT motion mode status

<b>BIT</b>	<b>VALUE</b>	<b>DESCRIPTION</b>
15	0	PVT buffer is not empty
	1	PVT buffer is empty – there is no PVT point in the buffer and the execution of the current PVT segment is over. If <b>PVTSENOFF</b> = 0 (default), the drive/motor will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
14	0	PVT buffer is not low
	1	PVT buffer is low – the number of PVT points from the buffer is equal or less than the low limit set using <b>SETPVT</b> . If <b>PVTSENOFF</b> = 0 (default), the drive will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
13	0	PVT buffer is not full
	1	PVT buffer is full – the number of PVT points from the buffer is equal with the buffer dimension. If <b>PVTSENOFF</b> = 0 (default), the drive will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
12	0	No integrity counter error
	1	Integrity counter error. If integrity counter error checking is enabled and <b>PVTSENOFF</b> = 0 (default), the drive will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
11	0	The drive has kept the PVT motion mode after a PVT buffer empty condition, because the velocity of the last PVT point was 0
	1	The drive has performed a Quick stop, following a PVT buffer empty condition, because the velocity of the last PVT point was different from 0
10	0	Normal operation. Data received are PVT points
	1	A PT point was received while PVT mode is active. The PT point was discharged. If <b>PVTSENOFF</b> = 0 (default), the drive/motor will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
9..7	0	Reserved
6..0	0..127	Current integrity counter value

**Remarks:** The function block just sends the PVT points, the motion mode is programmed with function block *TS\_SetPVT*, and therefore the *TS\_PVTPoint* must follow *TS\_SetPVT* call.

---

### 3.6.9 FB TS\_SetPT

#### ST prototype:

TS\_SetPT(BOOL Execute, REAL PTPOS0, INT LowLevelValue, INT PVTCounterValue, UINT ControlWord BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
	Execute	BOOL	At rising edge the command is sent
	PTCounterValue	INT	The new counter value for the first PT point
	LowLevelValue	INT	The new value for low buffer signaling
	PTPOS0	REAL	The initial position for absolute PT mode
	ControlWord	WORD	Selects the block options
Input/Output	AxisDescription	Axis_TS	Axis information
	Done	BOOL	Signal that PT motion mode programmed successfully
	CommandAborted	BOOL	The motion command is aborted by another motion command
Output	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs a drive/motor to work in PT motion mode. In PT motion mode the drive/motor performs a positioning path described through a series of points. Each point specifies the desired **Position** and **Time**, i.e. contains a PT data. Between the points the built-in reference generator performs a linear interpolation.

**Remarks:** The function block just programs the drive/motor for PT mode, the motion mode is activated from function block TS\_PTPoint, and therefore the TS\_PTPoint must follow TS\_SetPT call. Also, the function block TS\_PTPoint sends the PT points to the drive/motor.

The PT motion mode can be started only when the previous motion is complete. However, you can switch at any moment to another motion mode by calling a different function block with

The PT mode can be relative (ControlWord.0 = 0) or absolute (ControlWord.0 = 1). In the absolute mode, each PT point specifies the position to reach. The initial position may be either the current position reference TML variable **TPOS** (ControlWord.12=1) or a preset value read from the TML parameter **PTPOS0** (ControlWord.12=0). In the relative mode, each PT point specifies the position increment relative to the previous point. In both cases, the time is relative to the previous point i.e. represents the duration of a PT segment. For the first PT point, the time is measured from the starting of the PT mode.

Each time when a new PT point is received it is saved into the PT buffer. The reference generator empties the buffer as the PT points are executed. The PT buffer is of type FIFO (first in, first out). The default length of the PT buffer is 7 PT points. Each entry in the buffer is made up of 9 words, so the default length of the PT buffer in terms of how much memory space is reserved is 63 (3Fh) words. The drive/motor automatically sends messages to the PLC when the buffer is full, low or empty. The messages contain the PT status (TML variable **PVTSTS**). The buffer full condition

occurs when the number of PT points in the buffer is equal with the buffer size. The buffer low condition occurs when the number of PT points in the buffer is less or equal with a programmable value. When ControlWord.7=1 the buffer low level is programmed with the value read from **LowLevelValue** input. The buffer empty condition occurs when the buffer is empty and the execution of the last PVT point is over. When the PT buffer becomes empty the drive/motor hen the PT buffer becomes empty the drive/motor keeps the position reference unchanged.

**Remark:**

1. The PVT and PT modes share the same buffer. Therefore the TML parameters and variables associated with the buffer management are the same.
2. Both the PT buffer size and its start address are programmable via TML parameters (*int@0x0864*) and PVTBUFLN (*int@0x0865*). Therefore if needed, the PT buffer size can be substantially increased. Use *TS\_WriteIntegerParameter* to change the PT buffer parameters.

On detecting a rising edge at **Execute** input the function block reads the inputs and sends the motion commands to the drive/motor. The output **Busy** is set and remains set until the function block **TS\_PTPoint** signals "motion mode active", moment when the **Done** output is set and **Busy** reset.

**Table 3.5.** ControlWord bits description

Bit	Value	Description
0	0	The PT mode is relative
	1	The PT mode is absolute
1	0	Target update mode 1. Generates new trajectory starting from the actual values of position and speed reference (i.e. don't update the reference values with load/motor position and speed)
	1	Target update mode 0. Generates new trajectory starting from the actual values of load/motor position and speed (i.e. update the reference values with load/motor position and speed)
7	0	No change in the buffer low parameter
	1	Change the buffer low parameter with the value specified in <b>LowLevelValue</b>
11-8	0	Reserved for new features
12	0	If PT mode is set as absolute, the initial position is taken from TML parameter PVTPOS0 (default = 0). The initial position is used to compute the distance to move up to the first PT point.
	1	If PT mode is set as absolute, the initial position is taken from TML variable <b>TPOS</b> . The initial position is used to compute the distance to move up to the first PT point.
13	0	No change to the internal integrity counter
	1	Change internal integrity counter with the value specified in <b>PTCounterValue</b>
14	0	Integrity checking is active
	1	Integrity checking inactive
15	0	Nothing
	1	Clear PVT buffer

---

**Remarks:**

1. *The function block requires the drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*
2. *If the application requires switching between discrete motion (position control) and continuous motion (speed control) the speed loop must be closed, also. In the **Drive setup** dialogue select **Position** at **Control Mode**, then enter the **Advanced** dialogue and close the speed loop. After the selection perform the tuning of the position controller and speed controller.*

---

### 3.6.10 FB TS\_PTPoint

#### ST prototype:

TS\_PTPoint(BOOL Execute, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, PTrRef PT\_REF, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input/Output	Execute	BOOL	At rising edge the command is sent
	PTREF	PT_Ref	Description of PT point
	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Signal the commanded motion is completed after the last point PT sent
	CommandAborted	BOOL	The motion command is aborted by another motion command
	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error message was received from the drive/motor while <b>!Done</b>
	ErrorID	WORD	Information about the error occurred
	PVTStatus	WORD	The PVT status send by the drive/motor

**Description:** The function block activates the PT motion mode and sends the PT points to a drive/motor previously programmed with function block TS\_SetPT to work in PT motion mode. In PT motion mode the drive/motor performs a positioning path described through a series of points. Each point specifies the desired **Position** and **Time**, i.e. contains a PT data. Between the points the built-in reference generator performs a linear interpolation.

The PTrRef structure contains the following elements:

Component	Data type	Description
Position	REAL.	Represents the position to be reached at the end of the PT segment
Time	UINT	Represents the time interval of the PT segment. The maximum value for time interval is 65535 IU.
Counter	INT	The counter value for the current point. The maximum value for the counter is 127.

On detecting a rising edge at **Execute** input the function block reads the inputs and sends the PT point to the drive/motor. The output **Busy** is set and remains set until the drive signals the motion is complete, moment when the **Done** output is set and **Busy** reset. If the last point sent to has non-zero velocity the drive/motor signals the condition quick stop and the function block sets the output **Error**.

The function block sends a PT point every time detects a rising edge at input **Execute**.

When the PLC receives a message from the drive/motor with PT status the function transfers its value to output **PTStatus**.

**Table 3.6** PT motion mode status

<b>BIT</b>	<b>VALUE</b>	<b>DESCRIPTION</b>
15	0	PT buffer is not empty
	1	PT buffer is empty – there is no PT point in the buffer and the execution of the current PT segment is over. If <b>PVTSENOFF</b> = 0 (default), the drive/motor will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
14	0	PT buffer is not low
	1	PT buffer is low – the number of PT points from the buffer is equal or less than the low limit set using <b>SETPT</b> . If <b>PVTSENOFF</b> = 0 (default), the drive will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
13	0	PT buffer is not full
	1	PT buffer is full – the number of PT points from the buffer is equal with the buffer dimension. If <b>PVTSENOFF</b> = 0 (default), the drive will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
12	0	No integrity counter error
	1	Integrity counter error. If the integrity counter error checking is enabled and <b>PVTSENOFF</b> = 0 (default), the drive will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
11	0	Reserved
10	0	Normal operation. Data received are PT points
	1	A PVT point was received while PT mode is active. The PVT point was discharged. If <b>PVTSENOFF</b> = 0 (default), the drive/motor will send the <b>PVTSTS</b> each time this bit goes from 0 to 1
9..7	0	Reserved
6..0	0..127	Current integrity counter value

**Remarks:** The function block just sends the PT points, the motion mode is programmed with function block **TS\_SetPT**, and therefore the **TS\_PTPoint** must follow **TS\_SetPT** call.

---

### 3.6.11 FB TS\_Homing

#### ST prototype:

TS\_Homing(BOOL Execute, REAL HomePosition, REAL HomeSpeed, UINT HomingAddress, BOOL Done, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input/Output	Execute	BOOL	Send the stop command at rising edge
	HomePosition	REAL	The position set at the end of the homing procedure
	HomeSpeed	REAL	The speed for searching the home condition
	HomingAddress	WORD	Program memory address of the homing procedure
	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	The homing procedure ended successfully
	Busy	BOOL	The function is waiting for homing procedure to end
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

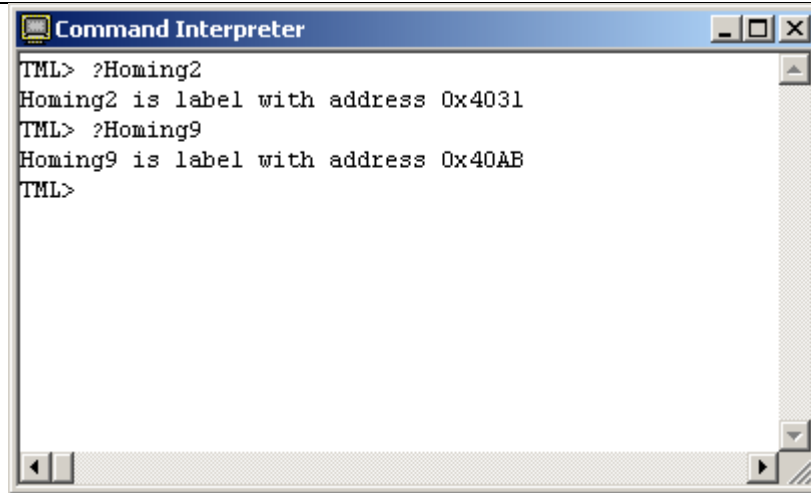
**Description:** The function block starts a homing procedure programmed on the drive/motor defined in structure **Axis**.

The search for the home position can be done in numerous ways. In order to offer maximum flexibility, the TML does not impose the homing procedures but lets you define your own, according with your application needs.

The homing procedures are programmed and downloaded on the drive/motor using **EasyMotion Studio** tool. Technosoft provides for each intelligent drive/motor a collection of up to 32 homing procedures. You may use any of these homing procedures as they are, or use them as a starting point for your own homing routines.

Steps to program the drive/motor for homing:

- Start **EasyMotion Studio** and load the project associated to your application
- Select from the list with all the defined homing procedures one or several to be used
- Select the menu command **Application | Motion | Build** to compile and link the TML program, the result is a file with out extension ready to be downloaded to the drive/motor.
- Establish the communication with your drive and download the TML program with menu command **Application | Motion | Download Program**
- Open the **Command interpreter** tool window and interrogate the drive/motor about the program memory address of the homing procedure using the command “**?HomingXX**” where **XX** is the homing procedure number.



```
Command Interpreter
TML> ?Homing2
Homing2 is label with address 0x4031
TML> ?Homing9
Homing9 is label with address 0x40AB
TML>
```

- In the PLC program call TS\_Homing with the address of homing procedure and the home position.

On detecting a rising edge at the input **Execute**, the function block sends the home position and starts the homing procedure on the drive/motor. The **Busy** output is set and remains set until the drive/motor signals homing procedure completed. When the drive/motor message is received the function block sets the **Done** output and resets **Busy**. Outputs **Error** and **ErrorID** have no associated functionality.



---

### 3.6.12 FB TS\_Stop

#### ST prototype:

TS\_Stop(BOOL Execute, REAL Deceleration, BOOL Done, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Send the stop command at rising edge
	Deceleration	REAL	Deceleration rate expressed in TML acceleration internal units.
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Signal drive velocity zero.
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block stops the drive/motor motor with the deceleration rate set in **Deceleration** input. The drive/motor decelerates following a trapezoidal position or speed profile. The stop command is sent to the drive/motor defined in structure **AxisDescription**.

On detecting a rising edge at **Execute** input the function block sends the stop command and sets the **Busy** output. **Busy** remains set until the drive/motor signals "velocity zero", moment when the function block sets the **Done** output and resets the **Busy**.

If the drive/motor error register is received, while the output **Done** is not set, the output Error is set and the

When the function block is called the axis is transferred to state **Stopping** and aborts the current motion function block. In the **Stopping** state no motion function block can be called.

The axis remains in state **Stopping** until the input **Execute** is reset when the axis is transferred to state **Standstill**.

If a drive's motion error register information is received during the execution of the function block, the output **Error** is set and its value is passed to **ErrorID**.

### 3.6.13 FB TS\_ExternalAnalogue

#### ST prototype:

TS\_ExternalAnalogue(BOOL Execute, BOOL PositionControl, BOOL SpeedControl, BOOL TorqueControl, REAL LimitValue, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
	Execute	BOOL	At rising edge the command is sent
Inputs	PositionControl	BOOL	The analogue reference is a position reference
	SpeedControl	BOOL	The analogue reference is a speed reference
	TorqueControl	BOOL	The analogue reference is a torque reference
	Limit value	REAL	Speed/acceleration limit value for position/speed control
Input/Output	AxisDescription	Axis_TS	Axis information
	Done	BOOL	Motion commands sent successfully
Output	CommandAborted	BOOL	The function block is aborted by another motion function block
	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error has occurred in the execution of the function block.
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs the drive/motor to work with an external analogue reference read via a dedicated analogue input (10-bit resolution). The analogue signal can be interpreted as a position, speed or torque analogue reference. Through inputs **PositionControl**, **SpeedControl** and **TorqueControl** you select the control type performed by the drive/motor. Table 3.1 shows the correspondence between the control type, inputs and the axis state.

**Remark:** During the drive/motor setup, in the **Drive setup** dialogue, you have to:

1. Select the appropriate control type for your application at Control Mode.
2. Perform the tuning of controllers associated with the selected control mode.
3. Setup the analogue reference. You specify the reference values corresponding to the upper and lower limits of the analogue input. In addition, a dead-band symmetrical interval and its center point inside the analogue input range may be defined.

**Table 3.7** Control type-Axis state

Control type	Input	Axis state
Position	PositionControl	DiscreteMotion
Speed	SpeedControl	ContinuousMotion
Torque	TorqueControl	ContinuousMotion

On detecting a rising edge at the **Execute** input, the function block checks the inputs **PositionControl**, **SpeedControl**, **TorqueControl** and starts sending the motion commands according with the **first input found set**. The output **Busy** is set and remains set until the last motion command is sent, moment when the function block sets the **Done** output and resets **Busy**.

---

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor.

In position control you can limit the maximum speed at sudden changes of the position reference and thus to reduce the mechanical shocks. In speed control you can limit the maximum acceleration at sudden changes of the speed reference and thus to get a smoother transition. These features are activated by setting the LimitValue to a non zero value.

---

### 3.6.14 FB TS\_ExternalDigital

#### ST prototype:

TS\_ExternalAnalogue(BOOL Execute, REAL LimitValue, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	At rising edge the command is sent
	LimitValue	REAL	Acceleration limit value
Input/Output	AxisDescription	Axis_TS	Axis information
	Done	BOOL	The motion command send successfully
Output	CommandAborted	BOOL	The motion command is aborted by another motion command
	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs the drive/motor to work with an external digital reference provided as pulse & direction or quadrature encoder signals. In either case, the drive/motor performs a position control with the reference computed from the external signals.

#### Remarks:

1. The function block requires the drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.
2. If the application requires switching between discrete motion (position control) and continuous motion (speed control) the speed loop must be closed, also. In the **Drive setup** dialogue select **Position** at **Control Mode**, then enter the **Advanced** dialogue and close the speed loop. After the selection perform the tuning of the position controller and speed controller.
3. The option for the input signals: pulse & direction or quadrature encoder is established during the drive/motor setup in External Reference Setup dialog.

On detecting a rising edge at the **Execute** input the function block starts sending the motion commands. The output **Busy** is set and remains set until the last motion command is sent, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor.

You can limit the maximum acceleration at sudden changes of the external reference and thus to get a smoother transition. This feature is activated by setting the **LimitValue** to a non zero value.

---

### 3.6.15 FB TS\_ExternalOnLine

#### ST prototype:

TS\_ExternalOnLine(BOOL Execute, BOOL PositionControl, BOOL SpeedControl, BOOL TorqueControl, REAL LimitValue, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameter description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	PositionControl	BOOL	The external reference is a position reference
	SpeedControl	BOOL	The external reference is a speed reference
	TorqueControl	BOOL	The external reference is a torque reference
	VoltageControl	BOOL	The external reference is a voltage reference
Input/Output	LimitValue	REAL	Speed/acceleration limit value for position/speed control
	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	The motion commands sent successfully
	CommandAborted	BOOL	The motion command is aborted by another motion command
	Busy	BOOL	The function block is processing the command
	Error	BOOL	Signal if an error has occurred in the execution of the function block or an error message was received from the drive
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs the drive/motor to work with an external reference read via a communication channel. Depending on the control mode chosen, the external reference is saved in one of the TML variables:

- **EREFP**, which becomes the position reference if the input **PositionControl** is set
- **EREFS**, which becomes the speed reference if the input **SpeedControl** is set
- **EREFT**, which becomes the torque reference if the input **TorqueControl** is set
- **EREFV**, which becomes voltage reference if the input **VoltageControl** is set

**Remark:** During the drive/motor setup, in the **Drive setup** dialogue, you have to:

1. Select the appropriate control type for your application at Control Mode.
2. Perform the tuning of controllers associated with the selected control mode.

When the external reference is sent from the PLC use:

- Function block **TS\_WriteLongParameter** to update the variable **EREFP** (long@0x02A8)
- Function block **TS\_WriteFixedParameter** to update the variable **EREFS** (fixed@0x02A8)
- Function block **TS\_WriteIntegerParameter** to update the variable **EREFT** or **EREFV** (int@0x02A9)

---

**Table 3.8** Control type-Axis state

<b>Control type</b>	<b>Input</b>	<b>Axis state</b>
Position	PositionControl	DiscreteMotion
Speed	SpeedControl	ContinuousMotion
Torque	TorqueControl	ContinuousMotion
Voltage	VoltageControl	ContinuousMotion

On detecting a rising edge at the **Execute** input, the function block checks the inputs **PositionControl**, **SpeedControl**, **TorqueControl**, **VoltageControl** and starts sending the motion commands according with the **first input found set**. The output **Busy** is set and remains set until the last motion command is sent, moment when the function block sets the **Done** output and resets **Busy**.

The **CommandAborted** output is set if another motion function block sends motion commands to the same drive/motor.

In position control you can limit the maximum speed at sudden changes of the position reference and thus to reduce the mechanical shocks. In speed control you can limit the maximum acceleration at sudden changes of the speed reference and thus to get a smoother transition. These features are activated by setting **LimitValue** to a non zero value.

---

### 3.6.16 FB TS\_GearIn

#### ST prototype:

TS\_GearIn(BOOL Execute, INT RatioNumerator, INT RatioDenominator, UDINT MasterResolution, REAL AccelerationLimit, BOOL InGear, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS Master, Axis\_TS Slave);

#### Parameter description:

	Parameter	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	RatioNumerator	INT	Gear ratio numerator (negative or positive)
	RatioDenominator	INT	Gear ratio denominator (only positive)
	MasterResolution	DINT	Master's position sensor resolution
	AccelerationLimit	REAL	Acceleration limit when the slave is coupling
	ControlWord	WORD	Selects the block options
Input/Output	Master	Axis_TS	Master axis information
	Slave	Axis_TS	Slave axis information
Output	InGear	BOOL	Is set when the gear ratio reached
	CommandAborted	BOOL	The function block is aborted by another function block
	Busy	BOOL	The function block is waiting the slave to reach the gear ratio
	Error	BOOL	Is set if drive's/motor's error register is received while <b>InGear</b>
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs a drive/motor to operate as slave in electronic gearing. In electronic gearing slave mode the drive/motor performs a position control. At each slow loop sampling period, the slave computes the master's position increment and multiplies it with its programmed gear ratio. The result is the slave's position reference increment, which added to the previous slave position reference gives the new slave position reference.

The gear ratio is a fixed value containing the result of the division **RatioNumerator** / **RatioDenominator**. **RatioNumerator** is a signed integer, while the **RatioDenominator** is unsigned integer.

**RatioNumerator** sign indicates the direction of movement: positive – same as the master, negative – reversed to the master. **RatioNumerator** and **RatioDenominator** are used by an automatic compensation procedure that eliminates the round off errors, which occur when the gear ratio is an irrational number like: 1/3 (Slave = 1, Master = 3).

The slave can get the master position in two ways:

1. Via a communication channel (ControlWord.5 = 0), from a drive/motor set as master with function block TS\_SetMaster
2. Via an external digital reference of type pulse & direction or quadrature encoder (ControlWord.5 = 1)

When master position is provided via the external digital interface, the slave computes the master position by counting the pulse & direction or quadrature encoder signals. The initial value of the

master position is set by default to 0. Use function block TS\_WriteLongParameter to change its value by writing the desired value in the TML variable APOS2 (long@0x091C).

**Remarks:**

1. The function block requires drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.
2. If the application requires switching between synchronized motion (position control) and continuous motion (speed control), i.e. use TS\_GearOut, you must also close the speed loop and perform the speed controller tuning.
3. Use function block **TS\_SetMaster** to program a drive/motor as master in electronic gearing
4. When the reference is read from second encoder or pulse & direction inputs you don't need to program a drive/motor as master in electronic gearing

You can smooth the slave coupling with the master, by limiting the maximum acceleration on the slave. This is particularly useful when the slave must couple with a master running at high speed. Setting ControlWord.7 and the maximum acceleration value **LimitValue** input activates the feature.

On detecting a rising edge at the **Execute** input the function block starts sending the motion commands. The output **Busy** is set and remains set until the drive/motor signals gear ratio reached, moment when the function block sets the **InGear** output and resets **Busy**.

If another motion function block starts sending motion commands while the output **InGear** is not set, the function block is aborted and the **CommandAborted** output is set. Also if the drive error register is received while the output **InGear** is not set, the output **Error** is set and the error register value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

During motion execution the axis is transferred into **SynchronousMotion** state and remains in this state until a new motion function block (except TS\_GearIn and TS\_CamIn) starts sending motion commands to the same drive/motor.

**Table 3.9** ControlWord bits description

Bit	Value	Description
0	0	Reserved
1	0	Enable operation as slave in electronic gearing
	1	Don't enable operation as slave in electronic gearing
2	0	Send gear ratio parameters
	1	Don't send the gear ratio parameters
3	0	Send the master resolution
	1	Don't send the master resolution
4	0	Use master resolution read from input MasterResolution
	1	Master resolution is 0x80000001 IU
5	0	Master position is received via communication channel
	1	Master position is read from 2 <sup>nd</sup> encoder or pulse & direction inputs
6	0	TUM0
	1	TUM1
7	0	The acceleration is not limited
	1	Limit the acceleration
8-15	0	Reserved for new features



---

### 3.6.17 FB TS\_GearOut

#### ST prototype:

TS\_GearOut(BOOL Execute, BOOL Slave, BOOL CommandAborted,, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS Slave);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Start disengaging process at rising edge
Input/Output	Slave	Axis_TS	Slave axis information
Output	Done	BOOL	Disengaging completed.
	Error	BOOL	Is set if drive's/motor's error register is received while <b>!Done</b>
	ErrorID	WORD	Information about the error occurred

**Description:** The function block disengages a slave drive/motor in electronic gearing from its master.

On detecting a rising edge at the input **Execute** the function block sends the motion commands for decoupling from the master and sets the **Busy** output. The **Done** output is set when the disengaging process was successfully executed.

The function block changes the slave axis state from SynchronizedMotion to ContinuousMotion, keeping the velocity from the moment when the function block was called.

If the drive's/motor's error register is received, while the output **Done** is not set, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

#### Remark:

*The function block requires the drive/motor position and speed loops to be closed. During the drive/motor setup select **Position** at Control Mode and in **Advanced** dialogue choose option **Close Position, Speed and Current loop**. After the selection perform the controllers tuning.*

---

### 3.6.18 FB MC\_CamTableSelect

#### ST prototype:

TS\_CamTableSelect(BOOL Execute, BOOL SlaveAbsolute, BOOL Done, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS Master, Axis\_TS Slave, ARRAY CamTable);;

#### Parameters description:

	Parameters	Data type	Description
Input	Execute	BOOL	Send motion commands at rising edge
	SlaveAbsolute	BOOL	Specifies the electronic camming type: Absolute = 1 Relative = 0
Input/Output	Master	Axis_TS	Master axis information
	Slave	Axis_TS	Slave axis information
	CamTable	ARRAY	Cam table reference
Output	Done	BOOL	Commanded position reached
	Busy	BOOL	Signal the function block is waiting for motion complete
	Error	BOOL	Signal if an error has occurred in the execution of the function block.
	ErrorID	WORD	Information about the error occurred

**Description:** The function block sets the cam table used by the slave drive/motor in electronic camming and the electronic camming type: absolute or relative. In the relative mode (SlaveAbsolute = 0), the output of the cam table is added to the slave actual position. in the absolute mode (SlaveAbsolute = 1), the output of the cam table it's the slave target position.

The cam tables are first downloaded into the EEPROM memory of the drive/motor, then, calling **MC\_CamTableSelect**, the selected cam table is copied from the EEPROM memory into the drive/motor RAM memory.

Through **CamTable** you select the active cam table. The **CamTable** is an array with two elements; the first represents the drive/motor EEPROM address where the cam table was downloaded, the second is the drive/motor RAM address where the cam table is copied. These addresses are drive/motor specific.

If you are using EasyMotion Studio to create and/or download the cam tables, in **Application General Information** open **Memory Settings** dialogue. If you have more than one cam table their addresses are obtained adding the cam length to the addresses reported in Memory Settings.

#### Remarks:

- *The function must be called before enabling the electronic camming*
- *During electronic camming slave mode, only one cam table can be active at time*

---

### 3.6.19 FB TS\_CamIn

#### ST prototype:

TS\_GearIn(BOOL Execute, REAL SlaveOffset, REAL MasterScaling, REAL SlaveScaling, DINT MasterResolution, REAL AccelerationLimit, BOOL InGear, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS Master, Axis\_TS Slave);

#### Parameter description:

	Parameter	Data type	Description
Input	Execute	BOOL	At rising edge the command is sent
	SlaveOffset	REAL	Cam table offset expressed in TML position units
	MasterScaling	REAL	CAM table input scaling factor
	SlaveScaling	REAL	CAM table output scaling factor
	MasterResolution	DINT	Master's position sensor resolution expressed in encoder counts
Input/Output	LimitValue	REAL	Speed limit value expressed in TML speed units
	ControlWord	UINT	Selects the block options
	Master	Axis_TS	Master axis information
Output	Slave	Axis_TS	Slave axis information
	InCam	BOOL	Signal the cam is engaged for the first time
	CommandAborted	BOOL	The motion command is aborted by another motion command
	Busy	BOOL	The function block is processing the command
	Error	BOOL	Is set if drive's/motor's error register is received while <b>!InCam</b>
	ErrorID	WORD	Information about the error occurred

**Description:** The function block programs a drive/motor to operate as slave in electronic camming. The slave drive/motor executes a cam profile function of the master drive/motor position. The cam profile is defined by a cam table – a set of (X, Y) points, where X is cam table input i.e. the master position and Y is the cam table output i.e. the corresponding slave position. Between the points the drive/motor performs a linear interpolation.

The slave can get the master position in two ways:

1. Via a communication channel (ControlWord.5 = 0), from a drive/motor set as master with function block TS\_SetMaster
2. Via an external digital reference of type pulse & direction or quadrature encoder (ControlWord.5 = 1)

When master position is provided via the external digital interface, the slave computes the master position by counting the pulse & direction or quadrature encoder signals. The initial value of the master position is set by default to 0. Use function block TS\_WriteLongParameter to change its value by writing the desired value in the TML variable APOS2 (long@0x091C).

#### Remarks:

1. *The function block requires drive/motor position loop to be closed. During the drive/motor setup select **Position** at Control Mode and perform the position controller tuning.*

2. If the application requires switching between synchronized motion (position control) and continuous motion (speed control), i.e. use MC\_CamOut, you must also close the speed loop and perform the tuning of the speed controller.
3. When the reference is read from second encoder or pulse & direction inputs you don't need to program a drive/motor as master in electronic camming

Through input **SlaveOffset** you can shift the cam profile versus the master position, by setting an offset for the slave. The cam table input is computed as the master position minus the cam offset. For example, if a cam table is defined between angles 100 to 250 degrees, a cam offset of 50 degrees will make the cam table to execute between master angles 150 and 300 degrees.

The electronic camming can be relative or absolute. In the relative mode, the output of the cam table is added to the slave actual position. In the absolute mode, the output of the cam table is the target position to reach, TML variable TPOS (long@0x02B2). The electronic camming mode is set with function block MC\_CamTableSelect

You can compress/extend the cam table input. Specify through input **MasterScaling** the correction factor by which the cam table input is multiplied. For example, an input correction factor of 2, combined with a cam offset of 180 degrees, will make possible to execute a cam table defined for 360 degrees of the master in the last 180 degrees.

You can also compress/extend the cam table output. Specify through input **SlaveScaling** the correction factor by which the cam table output is multiplied. This feature addresses the applications where the slaves must execute different position commands at each master cycle, all having the same profile defined through a cam table. In this case, the drive/motor is programmed with a unique normalized cam profile and the cam table output is multiplied with the relative position command updated at each master cycle.

On detecting a rising edge at the **Execute** input the function block starts sending the motion commands and sets the **Busy** output. The function block sends the motion commands according with the selections made through **ControlWord** input. **Busy** remains set until the drive/motor signals coupling completed, moment when the function block sets the **InCam** output and resets **Busy**.

While the output **InCam** is not set, the function checks:

- If other motion function blocks start sending commands to the same drive/motor. In this case the function is aborted and the output **CommandAborted** is set
- If the error register is received from the drive/motor. In this case the function block sets the output **Error**. The value of error register is transferred at the output **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

**Table 3.10.** ControlWord bits description

Bit	Value	Description
0	0	Send the cam table input scaling factor
	1	Don't send the cam table input scaling factor
1	0	Send the cam table input scaling factor
	1	Don't send the cam table input scaling factor
2	0	Send the cam table offset
	1	Don't send the cam table offset
3	0	Send the master resolution
	1	Don't send the master resolution

4	0	Use master resolution read from input MasterResolution
	1	Master resolution is 0x80000001
5	0	Master position is received via communication channel
	1	Master position is read from 2 <sup>nd</sup> encoder or pulse & direction inputs
6	0	Target Update Mode 1 (TUM1). Generates new trajectory starting from the actual values of position and speed reference
	1	Target Update Mode 0 (TUM0). Generates new trajectory starting from the actual values of load/motor position and speed
7	0	The slave speed is not limited in absolute electronic camming
	1	Limit the slave speed in absolute electronic camming
8-15	0	Reserved

---

### 3.6.20 FB MC\_CamOut

#### ST prototype:

TS\_CamOut(BOOL Execute, BOOL Done, BOOL CommandAborted, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS Slave);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Start disengaging process at rising edge
Input/Output	Slave	Axis_TS	Slave axis information
	Done	BOOL	Disengaging completed.
Output	CommandAborted	BOOL	The motion command is aborted by another motion command
	Error	BOOL	Is set if the axis goes to ErrorStop while <b>!Done</b>
	ErrorID	WORD	Information about the error occurred

**Description:** The function block decouples a slave drive/motor set in electronic camming from its master.

On detecting a rising edge at the input **Execute** the function block sends the motion commands for decoupling from the master and sets the **Busy** output. The **Done** output is set when the disengaging process was successfully executed.

The function block changes the slave state from **SynchronizedMotion** to **ContinuousMotion**, keeping the velocity from the moment when the function block was called.

If the drive's/motor's error register is received, while the output **Done** is not set, the output **Error** is set and its value is passed to **ErrorID**.

All outputs remain set until **Execute** input is reset, but at least for one block call.

#### **Remark:**

*The function block requires the drive/motor position and speed loops to be closed. During the drive/motor setup select **Position** at Control Mode and in **Advanced** dialogue choose option **Close Position, Speed and Current loop**. After the selection perform the controllers tuning.*

---

### 3.6.21 FB TS\_SetMaster

#### ST prototype:

TS\_SetMaster(BOOL Execute, DINT SyncPeriod, UINT ControlWord, BOOL Done, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS Master, Axis\_TS Slave);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Send the stop command at rising edge
	SyncPeriod	DINT	Time period for synchronization messages expressed in TML time internal units
	ControlWord	UINT	Selects the block options
Input/Output	Master	Axis_TS	Master axis information
	Slave	Axis_TS	Slave axis information
Output	Done	BOOL	Master configuration commands sent
	Busy	BOOL	The function is processing the commands
	Error	BOOL	Not used
	ErrorID	WORD	Not used

**Description:** The function block programs a drive/motor as master in electronic gearing or camming. The **Master** structure defines the master drive/motor, while the slave is defined in the **Slave** structure.

The master operation is enabled when the function block is called with ControlWord1=0 and disabled when the function block is called with ControlWord.0=1. In both cases, these operations have no effect on the motion executed by the master.

Once at each slow loop sampling time interval, the master sends either its load position (ControlWord.4 = 0) or its position reference (ControlWord.4=1) to the axis or the group of axes defined in the structure **Slave**.

When ControlWord.15=0 the **Slave** structure contains the axis ID of one slave. When ControlWord.15 =1 its content is interpreted as a group ID, i.e. the group of slaves to which the master should send its data.

**Remark:** You need to specify the Axis ID or the Group ID where master sends its position only the first time (after power on) when a drive is set as master. If the master mode is later on disabled (function block called with ControlWord.0 = 1), then enabled (function block called with ControlWord.1 = 0) again, there is no need to set again the Axis ID or the Group ID, as long as they remain unchanged. In this case, set the ControlWord.2 =1.

When ControlWord.5=1 you can change the synchronization procedure between the master and the slave axes. The synchronization procedure is enabled when ControlWord.6=0, the time interval between synchronization messages is read from SyncPeriod input, or disabled when ControlWord.6=1. Recommended starting value is 20ms. When synchronization procedure is active, the execution of the control loops on the slaves is synchronized with those of the master within a 10s time interval.

In electronic gearing if the master activation is done AFTER the slaves are set in electronic gearing mode and its position is different from zero, set ControlWord.3=0 when the function block

is called. This determines the master to send an initialization message to the slaves. In electronic camming the ControlWord.3 **must be set**.

On detecting a rising edge at the input **Execute** the function block sends the configuration messages to the drive/motor defined in the structure Master. The function block sends the motion commands according with the selections made through **ControlWord** input. The **Done** output is set when the last message is sent. Outputs **Error** and **ErrorID** have no associated functionality.

**Table 3.11** ControlWord bits description

Bit	Value	Description
0	0	Nothing
	1	Disable operation as master in electronic gearing/camming
1	0	Enable the operation as master in electronic gearing/camming
	1	Don't enable the operation as master in electronic gearing/camming
2	0	Send the slave axis/group ID
	1	Don't send the slave axis/group ID
3	0	Initialize the slaves with actual/reference position in electronic gearing
	1	Don't initialize the slaves with actual/reference position in electronic gearing
4	0	The master sends its actual position
	1	The master sends its reference position
5	0	Skip the synchronization procedure setup
	1	Change the synchronization procedure according with ControlWord.7
6	0	Enable the synchronization procedure with time interval read from SyncPeriod
	1	Disable the synchronization procedure
7-14	0	Reserved
15	0	The master send its position to the drive/motor with axis ID read from Slave structure
	1	The master send its position to drives/motors members of group ID read from Slave structure



---

### 3.6.22 FB TS\_Power

#### ST prototype:

TS\_Power(BOOL Execute, BOOL Status, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description	
Input	Enable	BOOL	Enable = 1 enables power stage, 0 disables power stage	
	AxisDescription	Axis_TS	Axis information	
Input/Output	Status	BOOL	Effective power stage state	
	Busy	BOOL	Signal the function is waiting for power stage status	
	Output	Error	BOOL	Signal if an error has occurred in the execution of the function block
		ErrorID	WORD	Information about the error occurred

**Description:** The function block enables/disables (AxisOn/AxisOff) the power stage of the drive/motor with axis ID read from structure **Axis**.

On detecting a rising edge at **Enable** input the function block enables the power stage. If no error occurs the axis is transferred from **Disable** state to **Standstill**. If the drive/motor responds with an error message the axis is transferred from **Disable** to **Errorstop**. The power stage of the drive/motor is disabled when the **Enable** input is reset and the axis is transferred to **Disable** state.

The output **Status** reflects the effective power stage state. If power fails, i.e. during operation the input **Enable** is set and the drive/motor power stage is disabled, the axis is transferred to **Errorstop** state, the **Error** output is set and **ErrorID** becomes equal with 0xFFFF.

**Remark:** The function *TS\_AxisInit* **must** be executed prior *MC\_Power* call, in order to initialize the automatic messages feature of the drive/motor from the selected axis.

---

### 3.6.23 FB TS\_Reset

#### ST prototype:

TS\_Reset(BOOL Execute, BOOL Done, BOOL Busy, BOOL Error, UINT ErrorID, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Reset the axis at the rising edge
	AxisDescription	Axis_TS	Axis information
Input/Output	Done	BOOL	The drive/motor reached StandStill state
	Busy	BOOL	The function block is waiting for drive/motor power stage to be re-enabled
	Error	BOOL	Signal if an error has occurred in the execution of the function block
	ErrorID	WORD	Information about the error occurred

**Description:** The function block makes the transition from **ErrorStop** to **Standstill** by resetting drive/motor FAULT status. The function can be called only from **ErrorStop** state.

A drive/motor enters in the FAULT status, when an error occurs. In the FAULT status:

- The drive/motor is in AXISOFF with the control loops and the power stage deactivated
- The TML program execution is stopped and all the TML functions called are cancelled
- The error register MER (uint@0x08FC) shows the type of errors detected and the status register SRH.15 signals the fault condition
- Ready and error outputs (if present) are set to the not ready level, respectively to the error active level. When available, ready green led is turned off and error red led is turned on.

**Remark:** The following conditions signaled in MER do not set the axis in **ErrorStop**:

- Command error
- Negative limit switch input on activate level
- Positive limit switch input on activate level
- Position wraparound
- Serial and CAN bus communication errors

On detecting a rising edge at the **Execute** input the function block sends the reset commands and sets the **Busy** output. **Busy** remains set until the drive/motor power stage is re-enabled, moment when the function block sets the **Done** output and resets the **Busy**. The axis is transferred to **Standstill** state.

If the power stage is not enabled the axis remains in **ErrorStop** and the **Error** output is set. The value 0x0001 is transferred to **ErrorID** output.

The function can be called only from ErrorStop state.

---

### 3.6.24 FB TS\_ResetDrive

#### ST prototype:

TS\_ResetDrive(BOOL Execute, BOOL Done, BOOL Busy, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Send the commands at rising edge
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Signal drive velocity zero.
	Busy	BOOL	The function waits for status register

**Description:** Function block makes the transition from **ErrorStop** to **Disable** by resetting the drive drive/motor defined in structure **Axis**.

After a reset operation the drive/motor requires a time period (typically 20 ms) for initialization during which the communication is not available. The timer used internally by the function block is read from input **TimerNumber**.

**Remark:** *The function block reinitializes the drive/motor with the setup data found in EEPROM. If you changed the values of drive/motor parameters during normal operation and want to use the same values after the reset call TS\_SaveParameters prior function block TS\_ResetDrive*

On detecting a rising edge at the **Execute** input the function block sends the reset commands and sets the **Busy** output. The function block must be followed by a delay to allow the drive/motor to complete the reset command. **Busy** remains set until the last message is sent, moment when the function block sets the **Done** output and resets the **Busy**. The axis is transferred to **Disable** state.

---

### 3.6.25 FB TS\_WriteLongParameter

#### ST prototype:

TS\_WriteLongParameter(BOOL Execute, UINT ParameterAddress, DINT Value, BOOL Done, BOOL Busy, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Write long parameter at the rising edge
	ParameterAddress	UINT	Data memory address where the value will be written
	Value	DINT	Value to be written
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Is set when the write data message is sent
	Busy	BOOL	The function block is processing the command

**Description:** The function block sends a “write long/ulong data” message to the drive/motor with axis ID read from **Axis** structure. When the drive/motor receives the message it will write the **Value** in the data memory location **ParameterAddress**.

The TML uses the following data types:

- int – 16-bit signed integer
- uint – 16-bit unsigned integer
- fixed – 32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part
- **long** – 32-bit signed integer
- **ulong** – 32-bit unsigned integer

The data type uint or ulong are reserved for the TML predefined data. The user-defined variables are always signed. Hence you may declare them of type: int, fixed or long.

Each TML data (parameters, variables or registers) has an associated address. This represents the address of the data memory location where the TML data exists. Address ranges for TML data are from 0x0200 to 0x03AF and from 0x0800 to 0x09FF. For user-defined variables the address range is between 0x03B0 and 0x03FF.

On detecting a rising edge at the **Execute** input, the function block sends the “write data” message and sets the **Busy** output. The **Busy** output remains set for one function block call, in the next call, the function block sets the **Done** output and resets **Busy**. The output **Done** remain set until **Execute** input is reset.

---

### 3.6.26 FB TS\_WriteFixedParameter

#### ST prototype:

TS\_WriteFixedParameter(BOOL Execute, UINT ParameterAddress, REAL Value, BOOL Done, BOOL Busy, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input	EN	BOOL	Enable function execution
	Execute	BOOL	Write fixed parameter with the rising edge
	ParameterAddress	UINT	Data memory address where the value will be written
	Value	REAL	Value to be written
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Is set when the write data message is sent
	Busy	BOOL	The function block is processing the command
	ENO	BOOL	Status of function block execution

**Description:** The function block sends a “write fixed data” message to the drive/motor with axis ID read from **Axis** structure. When the drive/motor receives the message it will write the **Value** (converted to 32-bit fixed-point) in the data memory location **ParameterAddress**.

The TML uses the following data types:

- int – 16-bit signed integer
- uint – 16-bit unsigned integer
- **fixed** – 32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part
- long – 32-bit signed integer
- ulong – 32-bit unsigned integer

The data type uint or ulong are reserved for the TML predefined data. The user-defined variables are always signed. Hence you may declare them of type: int, fixed or long.

Each TML data (parameters, variables or registers) has an associated address. This represents the address of the data memory location where the TML data exists. Address ranges for TML data are from 0x0200 to 0x03AF and from 0x0800 to 0x09FF. For user-defined variables the address range is between 0x03B0 and 0x03FF.

On detecting a rising edge at the **Execute** input, the function block converts the real value read from **Value** input to 32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part. After the conversion, the “write fixed data” message is sent and the **Busy** output is set. **Busy** will remain set for one function block call, in the next call, the function block sets the **Done** output and resets **Busy**. The output **Done** remain set until **Execute** input is reset.

---

### 3.6.27 FB TS\_WriteIntegerParameter

#### ST prototype:

TS\_WriteIntegerParameter(BOOL Execute, UINT ParameterAddress, DINT Value, BOOL Done, BOOL Busy, Axis\_TS AxisDescription);

#### Parameters description:

	Parameter	Data type	Description
Input	Execute	BOOL	Write the parameter with the rising edge
	ParameterAddress	UINT	Data memory address where the value will be written
	Value	INT	Value to be written at the specified address
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Is set when the write data message is sent
	Busy	BOOL	The function is processing the command

**Description:** The function block sends a “write integer data” message to the drive/motor with axis ID read from **Axis** structure. When the drive/motor receives the message it will write the **Value** in the data memory location **ParameterAddress**.

The TML uses the following data types:

- **int** – 16-bit signed integer
- **uint** – 16-bit unsigned integer
- **fixed** – 32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part
- **long** – 32-bit signed integer
- **ulong** – 32-bit unsigned integer

The data type **uint** or **ulong** are reserved for the TML predefined data. The user-defined variables are always signed. Hence you may declare them of type: **int**, **fixed** or **long**.

Each TML data (parameters, variables or registers) has an associated address. This represents the address of the data memory location where the TML data exists. Address ranges for TML data are from 0x0200 to 0x03AF and from 0x0800 to 0x09FF. For user-defined variables the address range is between 0x03B0 and 0x03FF.

On detecting a rising edge at the **Execute** input, the function block sends the “write integer data” message and sets the **Busy** output. **Busy** will remain set for one function block call, in the next call, the function block sets the **Done** output and resets **Busy**. The output **Done** remain set until **Execute** input is reset.

---

### 3.6.28 FB TS\_ReadActualPosition

**ST prototype:**

TS\_ReadActualPosition(BOOL Execute, UINT ParameterAddress, DINT Value, BOOL Done, BOOL Busy, Axis\_TS AxisDescription);

**Parameter description:**

	Parameter	Data type	Description
Input	Execute	BOOL	Request the value of actual position continuously while enabled
Input/Output	AxisDescription	Axis_TS	Axis information
	Done	BOOL	The actual position read successfully
Output	Busy	BOOL	The function block is waiting to receive the value of actual position
	Error	BOOL	Not used
	ErrorID	WORD	Not used
	Position	REAL	Actual position expressed in TML position internal units

**Description:** The function block requests the value of the actual position from the drive/motor with axis ID found in **Axis** structure.

On detecting a rising edge at the **Enable** input, the function block sends the request message and sets the **Busy** output. **Busy** remains set until the requested value is available, moment when, the function block sets the **Done** output and resets **Busy**. The value received is converted to real and transferred to output **Position**. The output **Done** remain set until **Enable** input is reset.

**Error** and **ErrorID** outputs are reserved for future development and have no associated functionality.

---

### 3.6.29 FB TS\_ReadLongParameter

#### ST prototype:

TS\_ReadLongParameter(BOOL Execute, UINT ParameterAddress, BOOL Done, BOOL Busy, DINT Value, Axis\_TS AxisDescription);

#### Parameter descriptions:

	Parameter	Data type	Description
Input	Execute	BOOL	Get parameter continuously while Enable = 1
	ParameterAddress	UINT	Data memory address from where the value will be read
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Is set when the requested value is available
	Busy	BOOL	The function block is waiting to receive the requested value
	Value	DINT	The requested variable content

**Description:** The function block requests the value of the TML variable/parameter with data memory **ParameterAddress**. The TML data can be of type long/ulong – 32-bit signed/unsigned integer. The request is sent to the drive/motor with axis ID found in **Axis** structure.

Each TML data (parameters, variables or registers) has an associated address. This represents the address of the data memory location where the TML data exists. Address ranges for TML data are from 0x0200 to 0x03AF and from 0x0800 to 0x09FF. For user-defined variables the address range is between 0x03B0 and 0x03FF.

On detecting a rising edge at the **Enable** input, the function block sends the request message and sets the **Busy** output. **Busy** remains set until the requested value is available, moment when, the function block sets the **Done** output and resets **Busy**. The value received is transferred to output **Value**. The output **Done** remain set until **Enable** input is reset.



---

### 3.6.30 FB TS\_ReadFixedParameter

#### ST prototype:

TS\_ReadFixedParameter(BOOL Execute, UINT ParameterAddress, BOOL Done, BOOL Busy, REAL Value, Axis\_TS AxisDescription);

#### Parameter descriptions:

	Parameter	Data type	Description
Input	Execute	BOOL	Get parameter continuously while Enable = 1
	ParameterAddress	DINT	Data memory address from where the value will be read
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Is set when the requested value is available
	Busy	BOOL	The function is waiting to receive the requested value
	Value	REAL	The requested variable content

**Description:** The function block requests the value of the TML variable/parameter with data memory **ParameterAddress**. The TML data can be of type fixed – 32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part. The request is sent to the drive/motor with axis ID found in **Axis** structure.

Each TML data (parameters, variables or registers) has an associated address. This represents the address of the data memory location where the TML data exists. Address ranges for TML data are from 0x0200 to 0x03AF and from 0x0800 to 0x09FF. For user-defined variables the address range is between 0x03B0 and 0x03FF.

On detecting a rising edge at the **Enable** input, the function block sends the request message and sets the **Busy** output. **Busy** remains set until the requested value is available, moment when, the function block sets the **Done** output and resets **Busy**. The value received is converted to real and transferred to output **Value**. The output **Done** remain set until **Enable** input is reset.

---

### 3.6.31 FB TS\_ReadIntegerParameter

#### ST prototype:

TS\_ReadIntegerParameter(BOOL Execute, UINT ParameterAddress, BOOL Done, BOOL Busy, DINT Value, Axis\_TS AxisDescription);

#### Parameters descriptions:

	Parameter	Data type	Description
Input	Execute	BOOL	Get parameter continuously while Enable = 1
	ParameterAddress	UINT	Data memory address from where the value will be read
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Is set when the requested value is available
	Busy	BOOL	The function is waiting to receive the requested value
	Value	INT	The requested variable content

**Description:** The function block requests the value of the TML variable/parameter with data memory **ParameterAddress**. The TML data can be of type fixed – 32-bit fixed-point data with the 16MSB for the integer part and the 16LSB for the fractionary part. The request is sent to the drive/motor with axis ID found in **Axis** structure.

Each TML data (parameters, variables or registers) has an associated address. This represents the address of the data memory location where the TML data exists. Address ranges for TML data are from 0x0200 to 0x03AF and from 0x0800 to 0x09FF. For user-defined variables the address range is between 0x03B0 and 0x03FF.

On detecting a rising edge at the **Enable** input, the function block sends the request message and sets the **Busy** output. **Busy** remains set until the requested value is available, moment when, the function block sets the **Done** output and resets **Busy**. The value received is transferred to output **Value**. The output **Done** remain set until **Enable** input is reset.

---

### 3.6.32 FC TS\_BuildMessage

The library's function blocks use internally the TS\_BuildMessage to build the CAN messages. No parameters are required from the user.

**Remark:** The TS\_BuildMessage function has 2 source files, TS\_BuildMessage\_11\_bit.st and TS\_BuildMessage\_29\_bit.st, one for each CAN-bus protocol supported by the library. The name of the source files are suffixed with the length of the CAN identifier for easier identification of the protocol supported, i.e. for TechnoCAN protocol (11-bit CAN identifier) the name of the source file will be TS\_BuildMessage\_11\_bit. Add the source file corresponding to the actual CAN protocol used in the Step7 project before compiling the TS\_BuildMessage function.

---

### 3.6.33 FC TS\_InterpretMessage

**ST prototype:**

TS\_ReadLongParameter(BOOL Execute, UINT ParameterAddress, BOOL Done, BOOL Busy, DINT Value, Axis\_TS AxisDescription);

**Parameter description:**

	Parameters	Data type	Description
Input	RxBufferAddress	CANBuffer	Pointers to CAN module registers
	NoAxes	USINT	Number of Technosoft drives/motors controlled from PLC
	AxesArray	UDINT	Pointer to the array with axes information
Input/Output	–	–	–
Output	CANStatus	BYTE	CAN module status returned by function FC111–CANRCV

**Description:** The function interprets CAN-bus messages and saves the data received in the array with axes data. The function should be called periodically (every PLC cycle) at the beginning of the main program so that the last information received be available, in the same PLC cycle, for the other function blocks.

**Remarks:**

*The function processes only one CAN message per PLC cycle.*

*The TS\_InterpretMessage function has 2 source files, TS\_InterpretMessage\_11\_bit.st and TS\_InterpretMessage\_29\_bit.st, one for each CAN-bus protocol. The name of the source files are suffixed with the length of the CAN identifier for easier identification of the supported protocol, i.e. for TMLCAN protocol (29-bit CAN identifier) the name of the source file will be TS\_InterpretMessage\_29\_bit. Add the source file corresponding to the actual CAN protocol used in the Step7 project before compiling the TS\_InterpretMessage function.*

---

### 3.6.34 FC TS\_InitAxis

**ST prototype:**

TS\_InitAxis(UINT PLCAxisID, BOOL Execute, BOOL Done, Axis\_TS AxisDescription);

**Parameter description:**

	Parameter	Data type	Description
Input	PLCAxisID	UINT	The address of the PLC in the CAN network
	Execute	BOOL	Enable function execution
Input/Output	AxisDescription	Axis_TS	Axis information
Output	Done	BOOL	Status of function execution

**Description:** The function enables the drive/motor to send messages automatically when its state changes. This feature is required for correct operation of the TML\_LIB\_X20.

The function must be executed every time you power on or reset the drive/motor.

---

### 3.6.35 FC TS\_SendCommand

**ST prototype:**

TS\_SendCommand(BOOL Execute, UINT OpCode, UDINT CommandDataW12, UDINT CommandDataW23, BOOL Done, Axis\_TS AxisDescription);

**Parameters description:**

	Parameter	Data type	Description
	Execute	BOOL	Send stop command at rising edge
	OpCode	UINT	TML command operation code
	CommandDataW12	UDINT	Data words 1 and 2 of TML command
	CommandDataW34	UDINT	Data words 3 and 4 of TML command
	Length	USINT	Number of data bytes
<b>Input/Output</b>	AxisDescription	Axis_TS	Axis information
<b>Output</b>	Done	BOOL	The TML command sent

**Description:** The function sends a TML command to the drive/motor defined in structure **Axis**.

With the rising edge of **Execute** the function block builds and sends a TML command from its binary code. The binary code can be found with Binary code viewer a tool integrated in EasyMotion Studio. The **Done** output is when the command is sent successfully



T E C H N O S O F T

