TML_LIB

v2.0

Motion Control Library for Technosoft Intelligent Drives

T E C H N O S O F T

User Manual

TECHNOSOFT

TML_LIB
v2.0
User Manual

P091.040.v20.UM.0609

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.

About This Manual

This book describes the motion library **TML_LIB v2.0**. The book is common for Microsoft Windows and Linux x86 versions of the library. The TML_LIB is a collection of functions, which can be integrated in a PC application. For Microsoft Windows version of the TML_LIB you can write the application in C#, C/C++, Delphi Pascal or Visual Basic. For Linux platforms the application for TML_LIB must be written in C/C++. With TML_LIB 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 PC. The TML_LIB allows you to communicate with Technosoft drive/motors via serial RS-232, RS-485, CAN-bus or Ethernet protocols.

Scope of This Manual

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

- **IDM240 / IDM640** (all models), with firmware **F000H/F250A/F251A** or later (revision letter must be equal or after H i.e. I, J, etc.)

- **IDM680** (all models), with firmware **F500A/F501A** or later

- **IDM3000** (all models), with firmware **F037K/F256A** or later

- **ISD720 / ISD860** (all models) with firmware **F000I/F250A** or later

- **ISCM4805 / ISCM8005** (all models), with firmware **F000H/F250A/F251A** or later

- **ISM4803** (all models), with firmware **F024I** or later

- **IBL3605 / PIM3605** (all models), with firmware **F020K/F253A/F254A** or later

- **IBL2403 / PIM2403** (all models), with firmware **F020H/F253A/F254A** or later

- **IBL2401 / PIM2401** (all models), with firmware **F020H/F253A/F254A** or later

- **IPS110** (all models), with firmware **F005H/F255A** or later

- **IPS210** (all models), with firmware **F005H/F255A** or later

- **IM23x** (models IS and MA), with firmware **F900H/F252A** or later

- **IS23x** (models MA), with firmware **F903H/F261A** or later

*IMPORTANT! For correct operation, these drives/motors must be programmed with one of the firmware revision listed above. **EasySetUp**[1] - Technosoft IDE for drives/motors setup, 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.*

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
- **FAxx** – firmware versions F000H, F020H, F024I, F005H, F900H, F250A, F251A or later
- **FBxx** – firmware versions F500A, F501A or later

Related Documentation

***Help of the EasyMotion Studio software platform** –* describes how to use the EasyMotion Studio, which support all new features added to revision H of firmware. It includes: motion system setup & tuning wizard, motion sequence programming wizard, testing and debugging tools like: data logging, watch, control panels, on-line viewers of TML registers, parameters and variables, etc.

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.

---

[1] **EasySetUp** is included in **TML_LIB** installation package as a component of **EasyMotion Studio Demo version**. It can also be downloaded free of charge from Technosoft web page

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

Contents

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 (EEPROM) 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 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 drives 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/C++, Delphi Pascal, Visual Basic or LabVIEW
3) Use a **PLCopen** compatible library with motion function blocks which can be integrated in a PLC application based on one of the IEC 61136 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 are done.

The **TML_LIB** library is part of option 2) – a collection of functions allowing you to implement motion control applications on a PC computer. The link between the Technosoft drives/motors and the PC can be done via serial link, via CAN-bus using a CAN interface or via Ethernet using an adapter/bridge between Ethernet and RS-232. Realized as a collection of high-level functions, the library allows you to focus on the main aspects related to your application specific implementation, and to simply use the drive and execute motion commands by calling appropriate functions from the library.

This manual presents how to install and use the components of the **TML_LIB** library.

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 PC*
- *EasyMotion Studio is also recommended if your application includes a homing as it comes with 32 predefined homing procedures to select from, with possibility to adapt them*

MyCode.C

```
…
TS_SelectAxis(5);
TS_MoveVelocity(12.5,,MOVE_IMMEDIATE, FROM_REFERENCE);
TS_SetEventOnTime(200, WAIT_EVENT, NO_STOP);
Int InValue = 0;
While(InValue == 0)   TS_GetInput(INPUT5, &InValue);
TS_Stop();
```

TML LIB
```
…
TS_MoveVelocity (…)
…
```

```
CSPD=12.5;
MODE SP;
UPD;
```

Communication channel
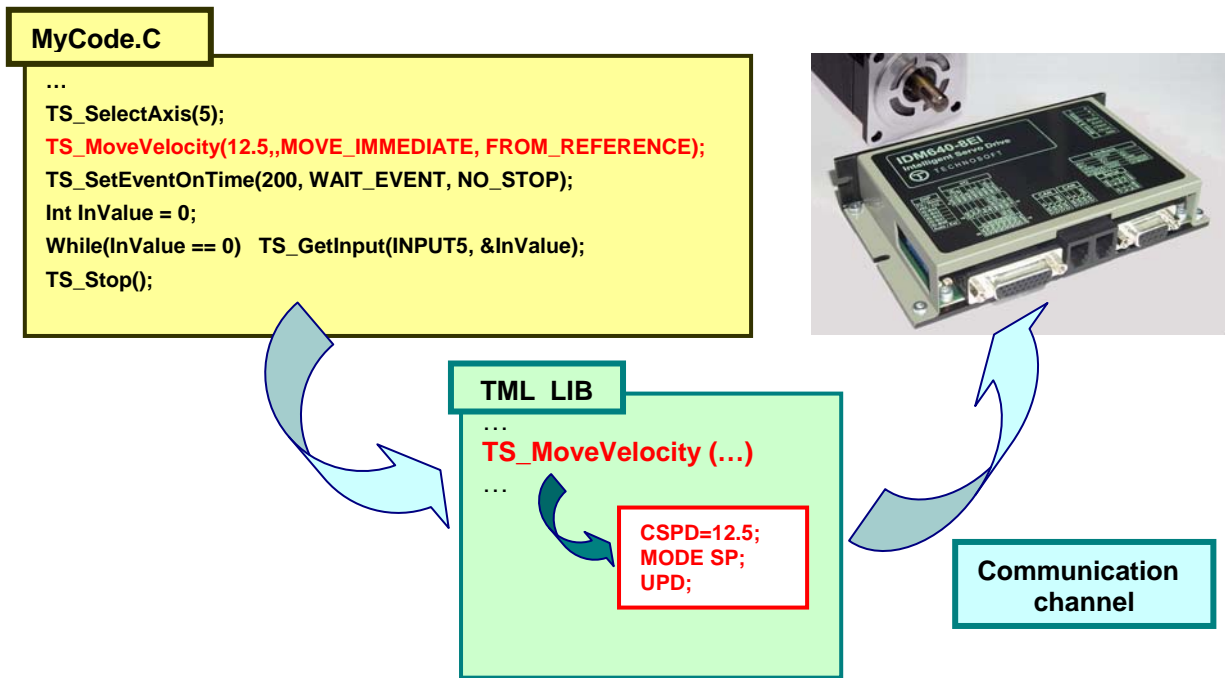
Figure 1.1. *Using TML_LIB to control a Technosoft intelligent drive from the PC computer*

2 Getting started

2.1 Hardware installation

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

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 one of the CAN interfaces supported by EasySetUp.

2.2 Software installation on Microsoft Windows platforms

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 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 library

Start the TML_LIB setup and follow the installation instructions. The package contents of the TML_LIB for Microsoft Windows is described in Appendix A.

*Remark: The Delphi application and the TML_lib.dll file must be in the same directory at run time. Hence, you have to copy the **TML_lib.dll** file in the Delphi project's folder (by default examples/DELPHIDemo) before running the application.*

2.3 Software installation on Linux x86 architectures

In order to perform successfully the following software installations, make sure that you have the "Root" rights and the following programs installed: **tar**, **gzip** and **sed**. Also, the TML_LIB library requires the GNU C library version 2 (**gcclib-2.***) and GNU Compiler Collection release 3 (**gcc-3.***).

2.3.1 Installing Microsoft Windows emulator

EasyMotion Studio and EasySetUp are applications build for Microsoft Windows operating systems. Hence to use the applications you must install an emulator for Microsoft Windows, for example **Wine**.

2.3.2 Installing EasySetUp

On the TML_LIB 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 using the Microsoft Windows emulator and follow the installation instructions.

2.3.3 Installing TML_LIB library

From the TML_LIB installation CD copy the file TML_lib_linux_x86.run. Change the file's access permissions with the command **chmod ugo +x TML_lib_linux_x86.run** and launch it. After you fill the registration information the library files will be automatically saved in the appropriate directories.

2.4 Build the host application with TML_lib

2.4.1 Drive/motor setup

Before starting to send motion commands from the PC, you need to do the drive/motor setup according with your application needs. For this operation you have to use **EasySetUp**, the integrated development environment for the configuration of the Technosoft drives and motors.

The output of **EasySetUp** is the setup table with all the information needed to configure and parameterize a Technosoft drive/motor. It must be downloaded to the drive/motor non-volatile memory. The setup table is copied at power-on into the RAM memory of the drive/motor and is used during runtime.

Steps for commissioning a Technosoft drive/motor

Step 1. Start EasySetUp

For Microsoft Windows platforms execute: "Start | Programs | EasySetUp | EasySetUp" or "Start | Programs | EasyMotion Studio | EasySetUp" depending on which installation package you have used. On Linux platforms use the Microsoft Windows emulator to start EasySetUp.

Step 2. Establish communication

Use the **Communication | Setup** command to check/change your PC communication settings. In the Communication Setup dialog select and configure the communication channel between the PC and the drive/motor. Press the **Help** button to find detailed information on how to setup the communication channels supported. Power on the drive/motor and then press the OK button to close the **Communication | Setup** dialog.

If the communication is established, then EasySetUp will display 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 informs you the error type. In this case, return to the Communication | Setup dialog, 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 table 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.

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

Step 6. Create the setup data for TML_LIB. The TML_LIB requires drive/motor setup information for the proper execution of the application. The setup data is generated with the **Setup | Export to TML_LIB…** command if you are in **EasySetUp**, or the **Application | Export to TML_LIB…** command if you are using EasyMotion Studio. The information is generated in the form of an archive file with the **.t.zip** extension and is saved in the **Archives** folder from EasySetUp/EasyMotion Studio installation folder (by default C:\Program Files\Technosoft\ESM\).

2.4.2   Build your application with TML_LIB

TML_LIB is a collection of high level functions, grouped in several categories and provided as the **TML_LIB.dll** file.

Most of these functions are of Boolean type, and return a '**True'** value if the execution of the function performed without any error (at PC level). If the function returns a '**False'** value, you can retrieve the error description by calling the function **TS_GetLastErrorText**.

Steps to build the host application with TML_LIB:

1. **Create the PC application's project.** Launch your development environment and create a new project. For details read the development environment online help.

    Remark: For Borland C++ projects the user must define a WINDOWS or WIN32 symbol in order to compile the C/C++ application.

2. **Setup the communication**. The host application is based on the communication between PC and Technosoft drives/motors thus it should begin with the communication channel setup. The communication channel is opened with the **TS_OpenChannel** function. At the end of the application you must close the communication channel with function **TS_CloseChannel**.

3. **Load setup configurations.** The setup information is required by the library functions in order to check if there are incompatibilities between the drive and the operation to be executed (as an example, avoiding issuing an "Output port" command to a port which is an input port on that drive). The setup data is generated by EasySetUp/EasyMotion

Studio based on your actual configuration. The setup information is in the form of archive files with the **.t.zip** extension. The .t.zip files are saved in the **Archives** folder from EasyMotion Studio/EasySetUp installation folder. The setup data of the drive/motor are declared using the **TS_LoadSetup** function in the PC application. The TS_LoadSetup has as argument the *.**t.zip** file. The function must be called for each axis controlled through TML_lib.

4. **Setup axis.** Each axis defined at PC level requires the setup information. The configuration setup is associated to an axis with function **TS_SetupAxis**.

5. **Select the active axis/group.** The messages sent from the PC address to one axis. Use function **TS_SelectAxis** to choose the messages destination. All further function calls, which send TML messages on the communication channel, will address the messages to this active axis.

6. **Program the motion for current axis.** Use the TML_LIB functions to program the motions required.

3 TML_LIB description

3.1 Basic concept

The Technosoft intelligent drives are programmable using the Technosoft Motion Language (TML). TML consists of a high-level set of codes allowing the user to parameterize and execute specific motion operations.

TML allows to:

- Configure the motion mode (profiles, contouring, gearing in multiple axes structures, etc.)
- Detect / specifically treat external signals as limit switches, captures
- Execute homing sequences
- Setup / start specific action on pre-defined motion events
- Synchronize multiple axes structures, by sending group commands
- etc.

The **TML_LIB** library is the tool that helps you to handle the process of motion control application implementation on a PC computer, at a high level, without the need to write / compile TML code.

A central element of the library is the communication kernel, which is responsible of correct opening of the communication channel (serial RS-232 or RS-485, CAN-bus or Ethernet), as well as of TML messages handling. This includes handling of the specific communication protocol, for each of these channels.

Consequently, each application you'll develop starts with the opening of the communication channel, i.e. calling the **TS_OpenChannel** function. The application must end with the **TS_CloseChannel** function call.

You'll be able to handle multiple-axis applications from the PC. Besides the drive/motor setup with EasySetUp or EasyMotion Studio, you'll also need to indicate some basic drive information for correct usage of the library functions. Thus, for each drive that is installed in the system, you'll need to execute the **TS_SetupAxis** function, indicating the axis ID and configuration setup. Such information will be used for some functions of the library, in order to check if there are incompatibilities between the drive and the operation to be executed (as an example, avoiding issuing an "Output port" command to a port which is an input port on that drive).

Note that besides setting-up individual axes, it is also possible to setup groups of axes (with the **TS_SetupGroup** function). This allows you to issue commands which will be received and executed simultaneously on all the axes initialized as belonging to that group.

Once all the axes are defined, the library allows you to select the active axis or group, using the **TS_SelectAxis**, or **TS_SelectGroup** function respectively. Consequently, all future commands that you'll execute after the selection of one axis or group will be addressed to that axis or group. You can change at any time in your program the active axis/group. Also, a command can be sent to all the axes fom the newtwork, by selecting the destination with the TS_SelectBroadcast function.

3.2 Multithread and multiprocess applications with TML_LIB

The TML_LIB library supports multithread applications developed under C/C++ and Delphi. Each thread created in your application has to setup the communication, the axes and program the motion commands. For details about threads see the documentation of your development environment.

Remarks:

1. *For multithread applications created for Microsoft Windows, under Visual C++, the communication module of TML_LIB library, the **tmlcomm.dll**, must be dynamically linked at **load-time**.*
2. *The examples included in the package use the single thread variant of the library with the exception of Ex08_PVT. The example Ex08_PVT is available only for C/C++ and Delphi.*
3. *Applications developed under Visual Basic **must** use the single thread variant of the TML_lib.*

The following example presents the basic steps for creating a multithread application using C run-time library and the Win32 API:

1. Include the header **<windows.h>** for all the Win32 specific thread information
2. Define an array of handles and an array of thread id's.
3. Declare structures for passing to the controlling functions of each thread (define here the parameters you might be interested on).
4. Define global pointers to the structures required.
5. Declare the control functions for each thread. In Win32, thread functions MUST be declared like this: **DWORD WINAPI <name>(LPVOID)**
6. In the main body of your application call the function to create and start thread (in our example **CreateThread** function that actually creates and begins the execution of the thread).See the documentation of your development enviroment for more details.
7. Wait until all threads are done. Use **WaitForMultipleObjects** function. Read the help associated to the API call "WaitForMultipleObjects".
8. Close the handles of the threads with the function: **CloseHandle**.

These steps where followed to create the Example 43. The example commands 2 Technosoft drives/motors to execute a 2-D motion profile described by several linear and circular segments. The application uses 2 separate threads for each axis, in which computes the necessary commands for the associated drive. The application requires the multithread variant of the TML_lib, installed by default in the "C:\Program Files\Technosoft\TML_LIB\lib-multithread" folder.

```c
//You must include <windows.h> for all the Win32 specific thread
//information.

#include <windows.h>

//An array of handles to threads and of thread id's must be defined

HANDLE hThread[2];
DWORD dwThreadId_X, dwThreadId_Y;

//Structure for passing to the controlling function

typedef struct _PVT_data_Y {
```

```
long trace_y[N_MAX];
//You can add here other parameters you might be interested on
...
} PVT_DATA_Y, *P_PVT_DATA_Y;

typedef struct _PVT_data_X {

long trace_x[N_MAX];
//You can add here other parameters you might be interested on
...
} PVT_DATA_X, *P_PVT_DATA_X;

//Global definitions (pointers to the structures defined above)
PVT_DATA_X p_PVT_X;
PVT_DATA_Y p_PVT_Y;

//In Win32, thread functions MUST be declared like this:
//DWORD WINAPI <name>(LPVOID)
//The first thread function

DWORD WINAPI ThreadProc_X( LPVOID lpParam )
{
//Local definitions for functions parameters

//Cast to pointer to the structure specific to this controlling
function

P_PVT_DATA_X p_PVT_X;
p_PVT_X = (P_PVT_DATA_X)lpParam;

/*Write your specific code for the second thread (first open the
communication channel, load the setup file from the directory
created by Easy Setup or Easy Motion, make the setup of the Axis 001
based on the file previously loaded, make the setup of the Group n,
make the setup of the broadcast).Be careful in using the same global
variable in the bought threads    */
//Close the communication channel
return TRUE;
}

//The second thread function

DWORD WINAPI ThreadProc_Y( LPVOID lpParam )
{

//Cast to pointer to the structure specific to this controlling
function
P_PVT_DATA_Y p_PVT_Y;
p_PVT_Y = (P_PVT_DATA_Y)lpParam;
...
return TRUE;
}

int main()
{
```

```
/*The thread is created and its execution begin by calling the
CreateThread Win32 API Function. Please read your help files for
more details. */

hThread[0] = CreateThread(
NULL,              //default security attributes
0,                 //use default stack size
ThreadProc_X,      // thread function
pp_PVT_X,          // argument to thread function
0,                 // use default creation flags
&dwThreadId_X);    // returns the thread identifier

if (hThread[0] == NULL)
{
ExitProcess(0);
}

hThread[1] = CreateThread(NULL,0,ThreadProc_Y,pp_PVT_Y,0,
&dwThreadId_Y)
if (hThread[1] == NULL)
{
ExitProcess(1);
}

//Wait until all threads have terminated.
WaitForMultipleObjects(2, hThread, TRUE, INFINITE);

//Close all thread handles upon completion.
CloseHandle(hThread[0]);
CloseHandle(hThread[1]);

return 0;
}
/*You must include <windows.h> for all the Win32 specific thread
information.*/

#include <windows.h>

//An array of handles to threads and of thread id's must be defined

HANDLE hThread[2];
DWORD dwThreadId_X, dwThreadId_Y;

//Structure for passing to the controlling function

typedef struct _PVT_data_Y {

long trace_y[N_MAX];
//You can add here other parameters you might be interested on
...
} PVT_DATA_Y, *P_PVT_DATA_Y;

typedef struct _PVT_data_X {

long trace_x[N_MAX];
//You can add here other parameters you might be interested on
```

```
...
} PVT_DATA_X, *P_PVT_DATA_X;

//Global definitions (pointers to the structures defined above)
PVT_DATA_X p_PVT_X;
PVT_DATA_Y p_PVT_Y;

/*In Win32, thread functions MUST be declared like this:
DWORD WINAPI <name>(LPVOID)
The first thread function*/
DWORD WINAPI ThreadProc_X( LPVOID lpParam )
{
//Local definitions for functions parameters

//Cast to pointer to the structure specific to this controlling
function

P_PVT_DATA_X p_PVT_X;
p_PVT_X = (P_PVT_DATA_X)lpParam;

/*Write your specific code for the second thread (first open the
communication channel, load the setup file from the directory
created by Easy Setup or Easy Motion, make the setup of the Axis 001
based on the file previously loaded, make the setup of the Group n,
make the setup of the broadcast).Be careful in using the same global
variable in the bought threads    */
//Close the communication channel
return TRUE;

}

//The second thread function
DWORD WINAPI ThreadProc_Y( LPVOID lpParam )
{

//Cast to pointer to the structure specific to this controlling
function

P_PVT_DATA_Y p_PVT_Y;
p_PVT_Y = (P_PVT_DATA_Y)lpParam;

...
return TRUE;

}
```

Depending on the communication channel used, the TML_LIB can share the communication resources enabling you to build multiprocess application. The communication devices suited for multiprocess applications are the serial interfaces (RS232 and RS485) and the CAN interfaces:

- from the Electronic System Design (ESD) – under Linux and Microsoft Windows
- from Peak System – under Linux

3.3 Functions descriptions

The section presents the functions implemented in the **TML_LIB** library. The functions are classified as follows:

- **Communication setup** – functions that manage the PC communication channel
- **Drive setup** – functions for axis setup in the PC application
- **Drive administration** – functions that control the destination axis of the messages sent from the host
- **Drive/motor monitoring** – functions for monitoring the drive/motor status
- **Error handling –** functions for FAULT state reset and drive reset
- **Motion programming –** functions for motion programming on the selected axis.
- **Motor commands –** functions to enable/disable the motor power stage, start/stop the motion, change the value of the motor position and current
- **Events** – functions for events programming and test
- **TML jumps and function calls** – functions which allows you to execute code downloaded in the drive/motor memory
- **I/O handling** – functions for read/write operations with drive/motor I/O ports
- **Data transfer** – functions for read/write operations from/to the drive/motor memory
- **Miscellaneous –** functions to send individual TML commands and to view the binary code of a TML command
- **Data logger** – functions for logger setup and data upload

For each function you will find the following information:

- The C prototype
- Description of the arguments
- A functional description
- Name of related functions
- Examples reference. The examples are listed in chapter 4.

3.3.1 Communication setup

3.3.1.1 TS_OpenChannel

Prototype:

INT TML_EXPORT TS_OpenChannel(LPCSTR pszDevName, BYTE btType, BYTE nHostID, DWORD baudrate);

Arguments:

	Name	Description
Input	pszDevName	The communication channel to be opened
	btType	The type of the communication channel and the CAN-bus communication protocol
	nHostID	Axis ID for the PC
	Baudrate	Communication baud rate
Output	Return	The file descriptor of the or -1 if error

Description: The function opens the communication channel specified with parameter **pszDevName**.

The **btType** parameter specifies the communication channel type and the CAN-bus communication protocol used by the application. **btType = ChannelType | ProtocolType**.

The TML_LIB supports the following types of communication channels:

- serial RS-232
    - ○ **ChannelType = CHANNEL_RS232** for PC serial port
    - ○ **ChannelType = CHANNEL_VIRTUAL_SERIAL** for communication through a user implemented serial driver. To properly interface the serial driver with the tmlcomm.dll, the user must follow the next steps:
        - a. initialize the communication channel with the serial settings implemented on the Technosoft drives/motors: 8 data bits, 2 stop bits, no parity, no flow control and one of the following baud rates: 9600 (default after reset), 19200, 38400, 56600 and 115200.
        - b. Implement the functions for interfacing the custom communication driver with the tmlcomm.dll. See the virtRS232.cpp file from the **virtRS232** example project.
        - c. Export the functions from the communication driver using a module-definition (.DEF) file. See the virtRS232.def file from the **virtRS232** example project.

- serial RS-485
    - ○ **ChannelType = CHANNEL_RS485** for an RS-485 interface board or an RS-232/RS-485 converter

- CAN-bus devices supported by TML_LIB for Microsoft Windows
    - ○ **ChannelType** = **CHANNEL_IXXAT_CAN** for **IXXAT** PC to CAN interface
    - ○ **ChannelType = CHANNEL_SYS_TEC_USBCAN** for **Sys Tec** USB to CAN interface

- o **ChannelType = CHANNEL_ESD_CAN** for **ESD** CAN interfaces
- o **ChannelType = CHANNEL_LAWICEL_USBCAN** for **Lawicel** CAN interface
- o **ChannelType = CHANNEL_PEAK_SYS_PCAN_PCI** for **PEAK System** PC-PCI to CAN interface
- o **ChannelType = CHANNEL_PEAK_SYS_PCAN_ISA** for **PEAK System** PCAN-ISA
- o **ChannelType = CHANNEL_PEAK_SYS_PCAN_PC104** for **PEAK System** PC/104
- o **ChannelType = CHANNEL_PEAK_SYS_PCAN_USB** for **PEAK System** USB to CAN interface
- o **ChannelType = CHANNEL_PEAK_SYS_PCAN_DONGLE** for **PEAK System** Dongle interfaces

- CAN-bus devices supported by TML_LIB for Linux
  - o **ChannelType = CHANNEL_ IXXAT _CAN** for **IXXAT** CAN interfaces supported by the Basic CAN interface from IXXAT
  - o **ChannelType = CHANNEL_ESD_CAN** for **ESD** CAN interfaces
  - o **ChannelType = CHANNEL_PEAK_SYS_PCAN_ISA** for **PEAK System** PCAN-ISA
  - o **ChannelType = CHANNEL_PEAK_SYS_PCAN_PC104** for **PEAK System** PC/104

*Remark: The TML_lib for Linux package contains a patch for drivers of the CAN-bus interfaces from **PEAK System** which enables the hardware filtering of the CAN messages. After TML_LIB installation, apply the patch, compile and install the driver using the drivers' documentation.*

- Ethernet[1]
  - o **ChannelType = CHANNEL_XPORT_IP** for Technosoft Ethernet to RS232 adapter

The CAN-bus communication protocols supported by the TML_LIB are:

- **ProtocolType = PROTOCOL_TMLCAN** – 29-bit CAN identifier
- **ProtocolType = PROTOCOL_TECHNOCAN** – 11-bit CAN identifier

Remarks:

1. *By default the TML_LIB uses the TMLCAN communication protocol, thus if your drive/ motor supports only TMLCAN protocol then the **btType = ChannelType**.*
2. *The specification of CAN-bus protocol is required when the PC is connected directly to the CAN-bus through a PC to CAN interface or in the cases when the drive/motor connected to the PC via RS232/Ethernet acts as a retransmission relay between the PC and the CAN-bus network. More details about the retransmission relay concept can be found in EasyMotion Studio on line help.*

Depending on the communication channel type, the parameter **pszDevName** can be:

- For serial communication:
  - o 'COM1', 'COM2', 'COM3'…. for Microsoft Windows version
  - o '/dev/ttyS0', '/dev/ttyS1', '/dev/ttyS3' for Linux version
- For virtual serial interface is the name of the dll file that implements the serial interface

---

[1] Supported only in the TML_LIB for Microsoft Windows

- For CAN-bus communication:
    - '1', '2', '3'… for Microsoft Windows version
    - '/dev/pcan0', '/dev/pcan8' for Linux version
- For Ethernet communication: '192.168.19.52', 'technosoft.masterdrive.ch'…

The **nHostID** parameter represents the Axis ID of the PC in the system. The value of **nHostId** is set as follows:

- For serial RS-232 the **nHostID** is equal with the axis ID of the drive connected to the PC serial port
- For serial RS-485 and CAN-bus the **nHostId** must be a unique value. ***Attention!*** *Make sure that all the drives/motors from the network have a different address*
- For Ethernet communication the **nHostID** is equal with the axis ID of the drive connected to the serial port of the Ethernet adapter.

Set the communication speed with the ***BaudRate*** parameter. The accepted values are:

- For serial communication and Ethernet: 9600, 19200, 38400, 56000 or 115200 bps.
- For CAN-bus: 125000, 250000, 500000, 1000000 bps

Remark: *You can open several communication channels but only one can be active in an application at one moment. You can switch between the communication channels with function TS_SelectChannel.*

Related functions: TS_CloseChannel, TS_SelectChannel

Associated examples: all

3.3.1.2 TS_SelectChannel

Prototype:

BOOL TML_EXPORT TS_SelectChannel(INT fd);

Arguments:

	Name	Description
Input	fd	The communication channel file descriptor
Output	return	TRUE if no error, FALSE if error

Description: The function selects as active the communication channel described by parameter **fd**. All commands send towards the drives/motors will use the selected communication channel.

Remarks:

1. *Use function TS_OpenChannel to open the communication channels*
2. *The function TS_SelectChannel is not required in applications with only one communication channel*

Related functions: TS_OpenChannel, TS_CloseChannel

Associated examples: –

3.3.1.3 TS_CloseChannel

Prototype:

void TML_EXPORT TS_CloseChannel(INT fd);

Arguments:

	Name	Description
Input	fd	The communication channel file descriptor
Output	–	–

Description: The function closes the communication channel described by parameter **fd**. With **fd = -1** the function closes the channel previously selected with function **TS_SelectChannel**. This function must be called at the end of the application. It will release the communication channel resources, as it was allocated to the program when the **TS_OpenChannel** function was called.

Related functions: TS_OpenChannel, TS_SelectChannel

Associated examples: all

3.3.2 Drive setup

3.3.2.1 TS_LoadSetup

Prototype:

INT TML_EXPORT TS_LoadSetup(LPCSTR setupDirectory);

Arguments:

	Name	Description
Input	setupDirectory	Name of the directory where are the setup files
Output	return	The index associated to the setup

Description: The function loads a drive/motor configuration setup in the PC application. The configuration setup is generated from EasyMotion Studio or EasySetUp and stored in two files: setup.cfg and variables.cfg. With string **setupDirectory** you specify the absolute or relative path of the directory with the setup files. The function returns an index associated to the configuration setup. Use the value returned to associate the configuration setup with the corresponding axis.

Remark: The function must be called for each configuration setup only once in your program, in the initialization part.

Related functions: TS_SetupAxis, TS_SetupGroup, TS_SetupBroadcast

Associated examples: all

3.3.2.2 TS_SetupAxis

Prototype:

BOOL TML_EXPORT TS_SetupAxis(BYTE axisID, INT idxSetup);

Arguments:

	Name	Description
Input	axisID	AxisID of the drive/motor
	idxSetup	Configuration index generated by TS_LoadSetup
Output	return	TRUE if no error, FALSE if error

Description: The function associates a configuration setup to the drive/motor having **axisID**. The configuration setup is identified through **idxSetup**.

The function must be called for each axis of the motion system, only once in your program, in the initialization part, before any attempt to send messages to that axis.

Remarks:

1. *The **axisID** parameter must be identical with the value set during drive/motor setup.*
2. *Use function TS_LoadSetup to obtain the configuration setup identifier.*

Related functions: TS_LoadSetup, TS_SetupGroup, TS_SetupBroadcast

Associated examples: all

3.3.2.3   TS_SetupGroup

Prototype:

BOOL TML_EXPORT TS_SetupGroup(BYTE groupID, INT idxSetup);

Arguments:

	Name	Description
Input	groupID	Group ID number. It must be a value between 1 and 8
	idxSetup	Name of the data file storing the setup axis information
Output	return	TRUE if no error, FALSE if error

Description: The function associates to the group of drives/motors a configuration setup identified through **idxSetup**. The configuration setup is used by TML_LIB when sends group commands.

The function must be called for each group defined in the motion system, only once in your program, in the initialization part, before any attempt to send messages to that group.

Remarks: Use function TS_LoadSetup to obtain the configuration setup identifier.

Related functions: TS_LoadSetup, TS_SetupAxis, TS_SetupBroadcast

Associated examples: –

3.3.2.4   TS_SetupBroadcast

Prototype:

BOOL TML_EXPORT TS_SetupBroadcast(INT idxSetup);

Arguments:

	Name	Description
Input	idxSetup	Name of the data file storing the setup axis information
Output	return	TRUE if no error, FALSE if error

Description: The function sets the configuration setup used by TML_LIB when issuing broadcast commands. The configuration setup is identified through **idxSetup**.

Remarks: Use function TS_LoadSetup to obtain the configuration setup identifier.

Related functions: TS_LoadSetup, TS_SetupAxis, TS_SetupGroup

Associated examples: Ex07_MultiAxes

3.3.2.5   TS_DriveInitialization

Prototype:

BOOL TML_EXPORT TS_DriveInitialization(void);

Arguments:

	Name	Description
Input	–	–
Output	Return	TRUE if no error, FALSE if error

Description: The function initializes the active axis. It must be executed when the drive/motor is powered or after a reset with function TS_Reset. The function call should be placed after the functions TS_SetupAxis and TS_SelectAxis and before any functions that send messages to the axis.

If the setup table is invalid then use the EasySetUp or EasyMotion Studio to download a valid setup. After setup table download the drive must be reset in order to activate the setup data.

Related functions: TS_LoadSetup, TS_SetupAxis, TS_SelectAxis

Associated examples: all

3.3.2.6 TS_Save

Prototype:

BOOL TML_EXPORT TS_Save(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error; FALSE if error

Description: The function saves the actual values of all the TML parameters with setup data from the active data RAM memory into the non-volatile memory, in the setup table. Through this command, you can save all the setup modifications done after the power on initialization.

Related functions: TS_Reset, TS_Save

Associated examples: –

3.3.3 Drive administration

3.3.3.1 TS_SelectAxis

Prototype:

BOOL TML_EXPORT TS_SelectAxis(BYTE axisID);

Arguments:

	Name	Description
Input	axisID	The axis ID where the commands are sent
Output	return	TRUE if no error, FALSE if error

Description: The function selects the currently active axis. All further function calls, which send TML messages on the communication channel, will address the messages to this active axis.

Call the function only after the setup of the axis (after calling the **TS_SetupAxis** function) for the same axis (with the same **AxisID**).

In a single axis motion system, call this function only once in your program. In a multiple axis configuration, call this function each time you want to redirect the communication to another axis of the system.

Related functions: TS_SelectGroup, TS_SelectBroadcast

Associated examples: all

3.3.3.2 TS_SelectGroup

Prototype:

BOOL TML_EXPORT TS_SelectGroup(BYTE groupID);

Arguments:

	Name	Description
Input	groupID	The group ID where the commands are sent
Output	return	TRUE if no error, FALSE if error

Description: The function selects the currently active group. All further function calls, which send TML messages on the communication channel, will address these messages to this active group. The active group is set with parameter **groupID**. It must be a value between 1 and 8.

*Remark: The function must be called after the group setup i.e. after calling the **TS_SetupGroup** function.*

Related functions: TS_SelectAxis, TS_SelectBroadcast

Associated examples: Ex08_PVT

3.3.3.3   TS_SelectBroadcast

Prototype:

BOOL TML_EXPORT TS_SelectBroadcast(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The function enables TML_LIB to issue the broadcast messages, i.e. all further function calls, which send TML messages on the communication channel, will address these messages to all the axes.

*Remark: The function must be called after the broadcast setup i.e. after calling the **TS_SetupBroadcast** function.*

Related functions: TS_SelectAxis, TS_SelectGroup

Associated examples: Ex07_MultiAxes

3.3.4 Drive/motor monitoring

3.3.4.1 TS_ReadStatus

Prototype:

BOOL TML_EXPORT TS_ReadStatus(SHORT SelIndex, WORD& Status);

Arguments:

	Name	Description
Input	SelIndex	Registers selection index
	Status	Pointer of the variable where the status is saved
Output	return	TRUE if no error; FALSE if error

Description: The function returns drive/motor status information. Depending on the value of **SelIndex** parameter, you can examine the contents of the Motion Control Register (**SelIndex = REG_MCR**), Motion Status Register (**SelIndex = REG_MSR**), Interrupt Status Register (**SelIndex = REG_ISR**), Status Register Low (**SelIndex = REG_SRL**), Status Register High (**SelIndex = REG_SRH**) or Motion Error Register (**SelIndex = REG_MER**) of the drive/motor.

Related functions: –

Associated examples: Ex02_DriveStatus, Ex03_ErrorHandling

3.3.4.2 TS_SendDataToHost

Prototype:

BOOL TML_EXPORT TS_SendDataToHost(BYTE HostAddress, DWORD StatusRegMask WORD ErrorRegMask);

Arguments:

	Name	Description
Input	HostAddress	The Axis ID of the host where the messages are sent
	StatusRegMask	Specifies the bits from status register that trigger the message
	ErrorRegMask	Specifies the bits from error register that trigger the message
Output	return	TRUE if no error, FALSE if error

Description: The function enables the active axis to send messages automatically to a host. The messages are triggered by conditions that change the drive/motor status or error register. The conditions are set trough parameters **StatusRegMask** and **ErrorRegMask**. The host Axis ID is set with parameter **HostAddress**.

Related functions: TS_RegisterHandlerForUnrequestedDriveMessages,
                   TS_CheckForUnrequestedDriveMessages

Associated examples: Ex02_DriveStatus

3.3.4.3 TS_CheckForUnrequestedDriveMessages

Prototype:

BOOL TML_EXPORT TS_CheckForUnrequestedDriveMessages(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The function checks if there are new unrequested messages received from the drive/motor. If the communication buffer contains an urequested message then it calls the user function that handles this type of messages. The function should be called periodically to have updated information.

Related functions: TS_RegisterHandlerForUnrequestedDriveMessages, TS_SendDataToHost

Associated examples: Ex02_DriveStatus

3.3.4.4   TS_RegisterHandlerForUnrequestedDriveMessages

Prototype:

**void TML_EXPORT TS_RegisterHandlerForUnrequestedDriveMessages
(pfnCallbackRecvDriveMsg handler);**

Arguments:

	Name	Description
Input	**pfnCallbackRecvDriveMsg**	Pointer to the user function
Output	–	–

Description: The function registers the user callback function that handles the unrequested messages sent by the drive/motor.

Related functions: TS_CheckForUnrequestedDriveMessages, TS_SendDataToHost

Associated examples: Ex02_DriveStatus

3.3.4.5 TS_OnlineChecksum

Prototype:

BOOL TML_EXPORT TS_OnlineChecksum(WORD startAddress, WORD endAddress WORD &checksum);

Arguments:

	Name	Description
Input	startAddress	The memory range start address
	endAddress	The memory range end address
	checksum	Pointer to the variable where the checksum is stored
Output	return	TRUE if no error, FALSE if error

Description: The function requests from the active axis the checksum of a memory range. The memory range is defined with parameters **startAddress** and **endAddress**. The function stores the checksum received from the drive in variable **checksum**.

With function TS_OnlineChecksum you can check the integrity of the data saved in a drive/motor non-volatile or RAM memory. The memory type is selected automatically function of the **startAddress** and the **endAddresses**.

Related functions: TS_SetBuffer

Associated examples: –

3.3.5 Error handling

3.3.5.1 TS_ResetFault

Prototype:

BOOL TML_EXPORT TS_ResetFault(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error; FALSE if error

Description: The function gets out the active axis from the FAULT status. A drive/motor enters in fault when an error occurs. After a TS_ResetFault execution, most of the errors bits from **M**otion **E**rror **R**egister are cleared (set to 0), the Ready output (if present) is set to the ready level, the Error output (if present) is set to the no error level and the drive/motor returns to normal operation.

Remarks:

- *The TS_ResetFault execution does not change the status of MER.15 (enable input on disabled level), MER.7 (negative limit switch input active), MER.6 (positive limit switch input active) and MER.2 (invalid setup table)*
- *The drive/motor will return to FAULT status if there are errors when the function is executed*

Related functions: TS_Power, TS_ReadStatus

Associated examples: Ex03_ErrorHandling

3.3.5.2 TS_Reset

Prototype:

BOOL TML_EXPORT TS_Reset(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error; FALSE if error

Description: The function resets the active axis. After reset the drive/motor will load the values of the TML parameters set during setup phase. If the drive/motor is configured to run in the '**Autorun'** mode, after reset, it will automatically execute the TML code stored in the E2ROM memory (if there is such a program).

Remark: If during drive/motor operation you have changed the setup parameters and want to use them after the reset, call function TS_Save prior TS_Reset. The function TS_Save stores the actual values of all TML parameters in the drive's/motor's non-volatile memory.

Related functions: TS_DriveInitialization, TS_Power, TS_DownloadProgram, TS_GOTO, TS_Save

Associated examples: Ex03_ErrorHandling.

3.3.5.3 TS_GetLastErrorText

Prototype:

LPCSTR TML_EXPORT TS_GetLastErrorText(void);

Arguments:

	Name	Description
Input	–	–
Output	return	A text related to the last occurred error

Description: The function returns the description of the last error occurred during the execution of a TML_LIB function.

Related functions: –

Associated examples: all

3.3.6 Motion programming

3.3.6.1 TS_MoveAbsolute

Prototype:

BOOL TML_EXPORT TS_MoveAbsolute(LONG AbsPosition, DOUBLE Speed, double Acceleration, SHORT MoveMoment, SHORT ReferenceBase);

Arguments:

	Name	Description
Input	AbsPosition	Position to reached expressed in TML position units
	Speed	Slew speed expressed in TML speed units. If the value is zero the drive/motor will use the previously value set for speed
	Acceleration	Acceleration/deceleration rate expressed in TML acceleration units. If its value is zero the drive/motor will use the previously value set for acceleration
	MoveMoment	Defines the moment when the motion is started
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
Output	return	TRUE if no error, FALSE if error

Description: The function programs an absolute positioning with trapezoidal speed profile. The motion is described through **AbsPosition** parameter for position to reach, **Speed** for slew speed and **Acceleration** for acceleration/deceleration rate. The position to reach can be positive or negative. The **Speed** and **Acceleration** can be **only** positive.

Once set, the motion parameters are memorized on the drive/motor. If you intend to use values previously defined for the acceleration rate and/or the velocity you don't need to send their values again in the following trapezoidal profiles. Set to zero the value of speed and/or acceleration and the drive/motor will use the values previously defined (this option reduces the TML code generated by this function).

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Set **ReferenceBase** = **FROM_REFERENCE** if you want the reference generator to compute the motion profile starting from the actual values of the position and speed reference. Set **ReferenceBase** = **FROM_MEASURE** if you want the reference generator to compute the motion profile starting from the actual values of the load/motor position and speed. When this option is used, at the beginning of each new motion profile, the position and speed reference are updated with the actual values of the load/motor position and speed.

Remark: In open loop control of steppers, this option is ignored because there is no position and/or speed feedback.

Related functions: TS_MoveRelative, TS_MoveSCurveAbsolute, TS_MoveSCurveRelative, TS_MoveVelocity.

Associated examples: Ex05_Homing, Ex07_MultiAxes, Ex11_IOHandling

3.3.6.2 TS_MoveRelative

Prototype:

BOOL TML_EXPORT TS_MoveRelative(LONG RelPosition, DOUBLE Speed, DOUBLE Acceleration, BOOL IsAdditive, SHORT MoveMoment, SHORT ReferenceBase);

Arguments:

	Name	Description
Input	RelPosition	Position increment expressed in TML position units
	Speed	Slew speed expressed in TML speed units. If its value is zero the drive/motor will use the previously value set for speed
	Acceleration	Acceleration/deceleration rate expressed in the TML acceleration units. If its value is zero the drive/motor will use the previously value set for acceleration
	IsAdditive	Specifies how is computed the position to reach
	MoveMoment	Defines the moment when the motion is started
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
Output	return	TRUE if no error, FALSE if error

Description: The function programs a relative positioning with trapezoidal speed profile. The motion is described through **RelPosition** for position increment, **Acceleration** for acceleration/deceleration rate and **Speed** for slew speed. The position increment can be positive or negative; the sign gives the motion direction. The speed and acceleration can be **only** positive.

Once set, the motion parameters are memorized on the drive/motor. If you intend to use values previously defined for the acceleration rate and/or the velocity you don't need to send their values again in the following trapezoidal profiles. Set to zero the value of speed and/or acceleration if you want the drive/motor to use the values previously defined with other commands (this option reduces the TML code generated by this function).

The position to reach can be computed in 2 ways: standard (default) or additive. In standard mode, the position to reach is computed by adding the position increment to the instantaneous position in the moment when the command is executed. In the additive mode, 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. The additive mode is activated with **IsAdditive = TRUE**.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Set **ReferenceBase** = **FROM_REFERENCE** if you want the reference generator to compute the motion profile starting from the actual values of the position and speed reference. Use this option

for example if successive standard relative moves must be executed and the final target position should represent exactly the sum of the individual commands. Set **ReferenceBase = FROM_MEASURE** if you want the reference generator to compute the motion profile starting from the actual values of the load/motor position and speed. When this option is used, at the beginning of each new motion profile, the position and speed reference are updated with the actual values of the load/motor position and speed.

Remark: In open loop control of steppers, this option is ignored because there is no position and/or speed feedback.

Related functions: TS_ MoveAbsolute, TS_MoveSCurveAbsolute, TS_MoveSCurveRelative, TS_MoveVelocity

Associated examples: Ex02_DriveStatus, Ex04_BasicMove, Ex05_Homing, Ex09_Logger, Ex10_EventHandling, Ex11_IOHandling

3.3.6.3 TS_MoveSCurveAbsolute

Prototype:

BOOL TML_EXPORT TS_MoveSCurveAbsolute(LONG AbsPosition, DOUBLE Speed, DOUBLE Acceleration, LONG JerkTime, SHORT MoveMoment, SHORT DecelerationType);

Arguments:

	Name	Description
Input	AbsPosition	Position to reach expressed in TML position units
	Speed	The slew speed expressed in TML speed units.
	Acceleration	Acceleration/deceleration rate expressed in TML acceleration units.
	JerkTime	Represents the time interval for acceleration to reach the programmed value. It is expressed in TML time units.
	MoveMoment	Defines the moment when the motion is started
	DecelerationType	Specifies the speed profile used when the motion is stopped with TS_Stop
Output	return	TRUE if no error, FALSE if error

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. The motion is described through **AbsPosition** parameter for position to reach, **Speed** for slew speed, **Acceleration** for acceleration/deceleration rate and **JerkTime**. The position to reach can be positive or negative. The **Speed**, **Acceleration** and **JerkTime** can be **only** positive.

An S-curve profile must begin when load/motor is not moving. During motion the parameters should not be changed. Therefore when executing successive S-curve commands, 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 TS_Stop, the deceleration phase can be done in 2 ways:

- Smooth, using an S-curve speed profile, when **DecelerationType = S_CURVE_SPEED_PROFILE**
- Fast, using a trapezoidal speed profile, when **DecelerationType = TRAPEZOIDAL_SPEED_PROFILE**

The motion can be executed:

- Immediately when **MoveMoment = UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment = UPDATE_ON_EVENT**
- If you select **MoveMoment = UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_MoveAbsolute, TS_MoveRelative, TS_MoveSCurveRelative TS_MoveVelocity, TS_QuikStopDecelerationRate

Associated examples: Ex04_BasicMove

3.3.6.4   TS_MoveSCurveRelative

Prototype:

BOOL TML_EXPORT TS_MoveSCurveRelative(LONG RelPosition, DOUBLE Speed, DOUBLE Acceleration, LONG JerkTime, SHORT MoveMoment, SHORT DecelerationType);

Arguments:

	Name	Description
Input	RelPosition	Position increment expressed in TML position units
	Speed	Slew speed expressed in TML speed units.
	Acceleration	Acceleration/deceleration rate expressed in TML acceleration units.
	JerkTime	Represents the time interval for acceleration to reach the programmed value. It is expressed in TML time units.
	MoveMoment	Defines the moment when the motion is started
	DecelerationType	Specifies the speed profile used when the motion is stopped with TS_Stop
Output	return	TRUE if no error, FALSE if error

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. The motion is described through **RelPosition** parameter for position increment, **Speed** for slew speed, **Acceleration** for acceleration/deceleration rate and **JerkTime**. The position to reach can be positive or negative. The **Speed**, **Acceleration** and **JerkTime** can be **only** positive.

An S-curve profile must begin when load/motor is not moving. During motion the parameters should not be changed. Therefore when executing successive S-curve commands, 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 TS_Stop, the deceleration phase can be done in 2 ways:

- Smooth, using an S-curve speed profile, when **DecelerationType = S_CURVE_SPEED_PROFILE**
- Fast, using a trapezoidal speed profile, when **DecelerationType = TRAPEZOIDAL_SPEED_PROFILE**

The motion can be executed:

- Immediately when **MoveMoment = UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment = UPDATE_ON_EVENT**
- If you select **MoveMoment = UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_MoveAbsolute, TS_MoveRelative, TS_MoveSCurveAbsolute TS_MoveVelocity

Associated examples: Ex05_BasicMove

3.3.6.5 TS_MoveVelocity

Prototype:

BOOL TML_EXPORT TS_MoveVelocity(DOUBLE Speed, DOUBLE Acceleration, SHORT MoveMoment, SHORT ReferenceBase);

Arguments:

	Name	Description
Input	Speed	Jog speed expressed in TML speed units
	Acceleration	Acceleration rate expressed in TML acceleration units. If the value is zero the drive/motor will use the previously value set for acceleration.
	MoveMoment	Defines the moment when the motion is started
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
Output	return	TRUE if no error, FALSE if error

Description: The function programs a trapezoidal speed profile. You specify the jog **Speed**. The load/motor accelerates until the jog speed is reached. The jog speed can be positive or negative; the sign gives the direction. The **Acceleration** can be **only** positive.

Once set, the motion parameters are memorized on the drive/motor. If you intend to use values previously defined for the acceleration rate you don't need to send its value again in the following speed profiles. Set to zero the value of acceleration if you want the drive/motor to use the value previously defined with other commands (this option reduces the TML code generated by this function).

The motion is executed:

- Immediately when **MoveMoment = UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment = UPDATE_ON_EVENT**
- If you select **MoveMoment = UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Set **ReferenceBase = FROM_REFERENCE** if you want the reference generator to compute the motion profile starting from the actual values of the position and speed reference. Use this option for example if successive standard relative moves must be executed and the final target position should represent exactly the sum of the individual commands. Set **ReferenceBase = FROM_MEASURE** if you want the reference generator to compute the motion profile starting from the actual values of the load/motor position and speed. When this option is used, at the beginning of each new motion profile, the position and speed reference are updated with the actual values of the load/motor position and speed.

Remark: In open loop control of steppers, this option is ignored because there is no position and/or speed feedback.

Related functions: TS_MoveRelative, TS_MoveAbsolute, TS_MoveSCurveAbsolute, TS_MoveSCurveRelative

---

Associated examples: Ex04_BasicMove, Ex06_ExternalReference, Ex07_MultiAxes, Ex10_EventHandling, Ex11_IOHandling

3.3.6.6 TS_SetAnalogueMoveExternal

Prototype:

BOOL TML_EXPORT TS_SetAnalogueMoveExternal(SHORT ReferenceType, BOOL UpdateFast, DOUBLE LimitVariation, SHORT MoveMoment);

Arguments:

	Name	Description
Input	ReferenceType	Specifies how the analogue signal is interpreted
	UpdateFast	Specifies how often the analogue reference is read when torque control is performed
	LimitVariation	Speed/acceleration limit value for position/speed control expressed in TML internal units
	MoveMoment	Defines the moment when the motion is started
Output	return	TRUE if no error, FALSE if error

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 parameter **ReferenceType** you specify how the analogue signal is interpreted:

- Position reference when **ReferenceType = REFERENCE_POSITION**. The drive/motor performs position control.
- Speed reference when **ReferenceType = REFERENCE_SPEED**. The drive/motor performs speed control.
- Torque reference when **ReferenceType = REFERENCE_TORQUE**. The drive/motor performs torque control.

*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.*

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 **LimitVariation** parameter to a positive value and disabled when the **LimitVariation** is zero.

In torque control you can choose how often to read the analogue input: at each slow loop sampling period (**UpdateFast = TRUE**) or at each fast loop sampling period (**UpdateFast = FALSE**).

The motion is executed:

- Immediately when **MoveMoment** = UPDATE_IMMEDIATE
- When a programmed event occurs if **MoveMoment** = UPDATE_ON_EVENT

- If you select **MoveMoment** = **UPDATE_NONE**, the motion parameters are set, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_SetDigitalMoveExternal, TS_SetOnlineMoveExternal

Associated examples: Ex06_ExternalReference

3.3.6.7 TS_SetDigitalMoveExternal

Prototype:

BOOL TML_EXPORT TS_SetDigitalMoveExternal(BOOL SetGearRatio, SHORT Denominator, SHORT Numerator, DOUBLE LimitVariation, SHORT MoveMoment);

Arguments:

	Name	Description
Input	SetGearRatio	Specifies if the digital reference is followed by the drive with a gear ratio
	Denominator	Gear ratio denominator
	Numerator	Gear ratio numerator
	LimitVariation	Acceleration limit value
	MoveMoment	Defines the moment when the motion is started
Output	return	TRUE if no error, FALSE if error

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: The option for the input signals: pulse & direction or quadrature encoder is established during the drive/motor setup.

The drive/motor follows the external reference with a gear ratio different than 1:1 when **SetGearRatio = TRUE**. The gear ratio is specified as a ratio of 2 integer values: **Numerator** / **Denominator**. The **Numerator** value is signed, while the **Denominator** is unsigned. The sign indicates the direction of movement: positive – same as the external reference, negative – reversed to the external reference.

You can limit the maximum acceleration at sudden changes of the external reference and thus to get a smoother transition. This feature is activated when the parameter **LimitValue** has a positive value and disabled when its value is zero.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the motion parameters are set, but the motion is not activated. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_SetAnalogueMoveExternal, TS_SetOnlineMoveExternal

Associated examples: Ex06_ExternalReference

3.3.6.8   TS_SetOnlineMoveExternal

Prototype:

BOOL TML_EXPORT TS_SetOnlineMoveExternal(SHORT ReferenceType, DOUBLE LimitVariation, DOUBLE InitialValue, SHORT MoveMoment);

Arguments:

	Name	Description
Input	ReferenceType	Specifies how the analogue signal is interpreted
	LimitVariation	Speed/acceleration limit value for position/speed control expressed in TML internal units
	InitialValue	The initial value of the reference received on-line
	MoveMoment	Defines the moment when the motion is started
Output	return	TRUE if no error, FALSE if error

Description: The function programs the drive/motor to work with a reference received via a communication channel from an external device. 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 **ReferenceType = REFERENCE_POSITION**
- **EREFS**, which becomes the speed reference if the **ReferenceType = REFERENCE_SPEED**
- **EREFT**, which becomes the torque reference if the **ReferenceType = REFERENCE_TORQUE**
- **EREFV**, which becomes voltage reference if the **ReferenceType = REFERENCE_VOLTAGE**

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

1. *Select the appropriate control type for your application in **Drive Setup** dialogue.*
2. *Perform the tuning of controllers associated with the selected control mode.*

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 **LimitVariation** parameter to a positive value and disabled when the **LimitVariation** is zero.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

If the external device starts sending the reference AFTER the motion mode is activated, it may be necessary to initialize EREFP, EREFS, EREFT or EREFV. The desired starting value is set through **InitialValue** parameter.

Related functions: TS_SetAnalogueMoveExternal, TS_SetDigitalMoveExternal

Associated examples: Ex06_ExternalReference

3.3.6.9 TS_VoltageTestMode

Prototype:

BOOL TML_EXPORT TS_VoltageTestMode(SHORT MaxVoltage, SHORT IncrVoltage, SHORT Theta0, SHORT Dtheta, SHORT MoveMoment);

Arguments:

	Name	Description
Input	MaxVoltage	Maximum test voltage expressed in TML voltage command units
	IncrVoltage	Voltage increment expressed in TML internal units
	Theta0	Initial value of electrical angle expressed in TML electrical angle units
	Dtheta	Electric angle increment expressed in TML electrical angle increment units
	MoveMoment	Defines the moment when the motion is started
Output	return	TRUE if no error, FALSE if error

Description: The function allows you to set the drives/motors in voltage test mode. In the test mode a saturated ramp voltage is applied to the motor, i.e. the voltage will increase with the **IncrVoltage** increment at each slow sampling period up to the **MaxVoltage** value.

Remark: This is a test mode to be used only in some special cases for drives setup. The test mode is not supposed to be used during normal operation

For AC motors (like for example the brushless motors), you have the possibility to rotate a voltage reference vector with a programmable speed. As a result, these motors can be moved in an "open-loop" mode without using the position sensor. The main advantage of this test mode is the possibility to conduct in a safe way a series of tests, which can offer important information about the motor parameters, drive status and the integrity of the its connections.

The voltage reference vector initial position is set through parameter **Theta0** and its speed through **Dtheta**. For DC motors set these parameters to zero.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_TorqueTestMode

Associated examples: –

3.3.6.10 TS_TorqueTestMode

Prototype:

BOOL TML_EXPORT TS_TorqueTestMode(SHORT MaxTorque, SHORT IncrTorque, SHORT Theta0, SHORT Dtheta, SHORT MoveMoment);

Arguments:

	Name	Description
Input	MaxTorque	Maximum test torque expressed in TML current units
	IncrTorque	Torque increment expressed in TML internal units
	Theta0	Initial value of electrical angle expressed in TML electrical angle units
	Dtheta	Electric angle increment expressed in TML electrical angle increment units
	MoveMoment	Defines the moment when the motion is started
Output	return	TRUE if no error, FALSE if error

Description: The function allows you to set the drives/motors in torque test mode. In the test mode a saturated ramp current is applied to the motor, i.e. the current will increase with the **IncrTorque** increment at each slow sampling period up to the **MaxTorque** value.

Remark: This is a test mode to be used only in some special cases for drives setup. The test mode is not supposed to be used during normal operation

For AC motors (like for example the brushless motors), you have the possibility to rotate a current reference vector with a programmable speed. As a result, these motors can be moved in an "open-loop" mode without using the position sensor. The main advantage of this test mode is the possibility to conduct in a safe way a series of tests, which can offer important information about the motor parameters, drive status and the integrity of the its connections.

The current reference vector initial position is set through parameter **Theta0** and its speed through **Dtheta**. For DC motors set these parameters to zero.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_VoltageTestMode

Associated examples: –

3.3.6.11 TS_PVTSetup

Prototype:

BOOL TML_EXPORT TS_PVTSetup(SHORT ClearBuffer, SHORT IntegrityChecking, SHORT ChangePVTCounter, SHORT AbsolutePositionSource, SHORT ChangeLowLevel, SHORT PVTCounterValue, SHORT LowLevelValue);

Arguments:

	Name	Description
Input	ClearBuffer	Specifies if the PVT buffer is cleared
	IntegrityChecking	Enable/disable PVT counter integrity checking
	ChangePVTCounter	Specifies if the integrity counter is updated with the value of PVTCounterValue parameter
	AbsolutePositionSource	Selects the source for initial position for absolute PVT mode
	ChangeLowLevel	Specifies if the level for BufferLow signaling is updated with the value of LowLevelValue parameter
	PVTCounterValue	The new value for the drive/motor PVT integrity counter
	LowLevelValue	The new value for the level of the BufferLow signal
Output	return	TRUE if no error, FALSE if error

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 with function TS_SendPVTFirstPoint and the PVT points are sent to the drive with function TS_SendPVTPoint.

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 or absolute. 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 (**AbsolutePositionSource = TRUE**) or a preset value read from the TML parameter PVTPOS0 (**AbsolutePositionSource = FALSE**). 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.

Remark: The PVT mode, absolute or relative, is set with function TS_SendPVTFirstPoint.

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 host 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. The level for BufferLow signaling is updated when **ChangeLowLevel = TRUE** with the value of parameter **LowLevelValue**. 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 PVTBUFBEGIN(int@0x0864) and PVTBUFLEN (int@0x0865). Therefore if needed, the PVT buffer size can be substantially increased. Use TS_SetIntegerVariable to change the PVT buffer parameters.*

Each PVT point also includes a 7-bit integrity counter. The integrity counter value must be incremented by the host by one, each time a new PVT point is sent to the drive/motor. If the integrity counter error checking is activated (**IntegrityChecking = TRUE)**, the drive compares its integrity counter value with the one sent with the PVT point. This comparison is done every time a PVT point is received. If the values of the two integrity counters do not match, the integrity check error is triggered, the drive/motor sends messages with PVTSTS to the host and the received PVT point is discarded. Each time a PVT point is accepted (the integrity counters match or the integrity counter error checking is disabled), the drive automatically increments its internal integrity counter.

The default value of the internal integrity counter after power up is 0. Set **ChangePVTCounter = TRUE** to change its value with **PVTCounterValue** parameter. The integrity counter checking is disabled when parameter **IntegrityChecking = FALSE**.

Related functions: TS_SendPVTFirstPoint, TS_SendPVTPoint

Associated examples: Ex08_PVT

3.3.6.12 TS_SendPVTFirstPoint

Prototype:

BOOL TML_EXPORT TS_SendPVTFirstPoint(LONG Position, DOUBLE Velocity, WORD Time, SHORT PVTCounter, SHORT PositionType, LONG InitialPosition, SHORT MoveMoment SHORT ReferenceBase);

Arguments:

	Name	Description
Input	Position	Position value for first PVT point expressed in TML position units
	Velocity	Speed at the end of the first PVT segment expressed in TML speed units
	Time	Represents the time interval of the PVT segment expressed in TML time units. The maximum time interval is 511 IU.
	PVTCounter	Integrity counter for first PVT point.
	PositionType	Specifies the type of PVT mode
	InitialPosition	The initial position at the start of an absolute PVT movement
	MoveMoment	Defines the moment when the motion is started
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
Output	return	TRUE if no error, FALSE if error

Description: The function sends the first PVT point and activates the PVT motion mode.

Parameter **PositionType** sets the PVT mode: absolute or relative. In the absolute mode (**PositionType = ABSOLUTE_POSITION**), each PVT point specifies the position to reach. The initial position may be either the current position reference TML variable TPOS or a preset value read from the TML parameter PVTPOS0. In the relative mode (**PositionType = RELATIVE_POSITION**), each PVT point specifies the position increment relative to the previous point.

Remark: The source for initial position, TPOS or PVTPOS0, is set with function TS_PVTSetup.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_PVTSetup, TS_SendPVTPoint

Associated examples: Ex08_PVT

3.3.6.13 TS_SendPVTPoint

Prototype:

BOOL TML_EXPORT TS_SendPVTPoint(LONG Position, DOUBLE Velocity, WORD Time, SHORT PVTCounter);

Arguments:

	Name	Description
Input	Position	Position at the end of the PVT segment expressed in TML position units
	Velocity	Velocity at the end of the PVT segment expressed in TML speed units
	Time	Time interval for the current PVT segment expressed in TML time units
	PVTCounter	The integrity counter for the current PVT point
Output	return	TRUE if no error, FALSE if error

Description: The function sends a PVT point to the drive/motor. Each point specifies the desired **Position**, **Velocity** and **Time**, i.e. contains a **PVT** data. Between the PVT points the reference generator performs a 3rd order interpolation. The PVT point also includes a 7-bit integrity counter. The integrity counter value must be incremented by the host by one, each time a new PVT point is sent to the drive/motor.

Related functions: TS_PVTSetup, TS_SendPVTFirstPoint

Associated examples: Ex08_PVT

3.3.6.14 TS_PTSetup

Prototype:

BOOL TML_EXPORT TS_PTSetup(SHORT ClearBuffer, SHORT IntegrityChecking, SHORT ChangePTCounter, SHORT AbsolutePositionSource, SHORT ChangeLowLevel, SHORT PTCounterValue, SHORT LowLevelValue);

Arguments:

	Name	Description
Input	ClearBuffer	When TRUE the PT buffer is cleared
	IntegrityChecking	Enable/disable PT counter integrity checking
	ChangePTCounter	Specifies if the integrity counter is updated with the value of PTCounterValue parameter
	AbsolutePositionSource	Selects the source for initial position for absolute PT mode
	ChangeLowLevel	Specifies if the level for BufferLow signaling is updated with the value of LowLevelValue parameter
	PTCounterValue	The new value for the drive/motor PT integrity counter
	LowLevelValue	The new value for the level of the BufferLow signal
Output	return	TRUE if no error, FALSE if error

Description: The function 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 **P**osition and **T**ime, i.e. contains a PT data. Between the points the built-in reference generator performs a linear interpolation.

Remark: The function block just programs the drive/motor for PT mode. The motion mode is activated with function TS_SendPTFirstPoint and the PT points are sent to the drive with function TS_SendPTPoint.

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.

The PT mode can be relative or absolute. 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 (**AbsolutePositionSource = TRUE**) or a preset value read from the TML parameter PVTPOS0 (**AbsolutePositionSource = FALSE**). 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.

Remark: The PT mode, absolute or relative, is set with function TS_SendPTFirstPoint.

Each time when the drive receives a new PT point, 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. The drive/motor automatically sends messages to the host 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. Set **ChangeLowLevel = TRUE** to change the level for BufferLow signaling with the value of parameter **LowLevelValue**. The buffer

empty condition occurs when the buffer is empty and the execution of the last PT point is over. When the PT buffer becomes empty the drive/motor keeps the position reference unchanged.

Remarks:

3. *The PT and PVT modes share the same buffer. Therefore the TML parameters and variables associated with the buffer management are the same.*
4. *Both the PT buffer size and its start address are programmable via TML parameters PVTBUFBEGIN(int@0x0864) and PVTBUFLEN (int@0x0865). Therefore if needed, the PT buffer size can be substantially increased. Use TS_SetIntegerVariabler to change the PT buffer parameters.*

Each PT point also includes a 7-bit integrity counter. The integrity counter value must be incremented by the host by one, each time a new PT point is sent to the drive/motor. If the integrity counter error checking is activated (**IntegrityChecking = FALSE)**, the drive compares its integrity counter value with the one sent with the PT point. This comparison is done every time a PT point is received. If the values of the two integrity counters do not match, the integrity check error is triggered, the drive/motor sends messages with PVTSTS to the host and the received PT point is discarded. Each time a PT point is accepted (the integrity counters match or the integrity counter error checking is disabled), the drive automatically increments its internal integrity counter.

The default value of the internal integrity counter after power up is 0. Set **ChangePTCounter = TRUE** to change the value of integrity counter with **PTCounterValue** parameter. The integrity counter checking is disabled when parameter **IntegrityChecking = TRUE**.

Related functions: TS_SendPTFirstPoint, TS_SendPTPoint

Associated examples: –

3.3.6.15 TS_SendPTFirstPoint

Prototype:

BOOL TML_EXPORT TS_SendPTFirstPoint(LONG Position, WORD Time, SHORT PTCounter, SHORT PositionType, LONG InitialPosition, SHORT MoveMoment SHORT ReferenceBase);

Arguments:

	Name	Description
Input	Position	Position value for first PT point expressed in TML position units
	Time	Time interval of the PT segment expressed in TML time units.
	PTCounter	Integrity counter for first PT point.
	PositionType	Specifies the type of PT mode
	InitialPosition	The initial position at the start of an absolute PT movement
	MoveMoment	Defines the moment when the motion is started
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
Output	return	TRUE if no error, FALSE if error

Description: The function sends the first PT point and activates the PT motion mode.

Parameter **PositionType** sets the PT mode: absolute or relative. In the absolute mode (**PositionType = ABSOLUTE_POSITION**), each PT point specifies the position to reach. The initial position may be either the current position reference TML variable TPOS or a preset value read from the TML parameter PVTPOS0. In the relative mode (**PositionType = RELATIVE_POSITION**), each PT point specifies the position increment relative to the previous point.

Remark: The initial position source, TPOS or PVTPOS0, is set with function TS_PTSetup.

The motion is executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_PTSetup, TS_SendPTPoint

Associated examples: –

3.3.6.16 TS_SendPTPoint

Prototype:

BOOL TML_EXPORT TS_SendPTPoint(LONG Position, WORD Time, SHORT PTCounter);

Arguments:

	Name	Description
Input	Position	Position at the end of the PT segment expressed in TML position units
	Time	Time interval for the current PT segment expressed in TML time units
	PTCounter	The integrity counter for the current PT point
Output	return	TRUE if no error, FALSE if error

Description: The function sends a PT point to the drive/motor. Each point specifies the desired **Position**, and **Time**. Between the PT points the reference generator performs a linear interpolation. The PT point also includes a 7-bit integrity counter. The integrity counter value must be incremented by the host by one, each time a new PT point is sent to the drive/motor.

Related functions: TS_PTSetup, TS_SendPTFirstPoint

Associated examples: –

3.3.6.17 TS_SetGearingMaster

Prototype:

BOOL TML_EXPORT TS_SetGearingMaster(BOOL Group, BYTE SlaveID, SHORT ReferenceBase, BOOL Enable, BOOL SetSlavePos, SHORT MoveMoment);

Arguments:

	Name	Description
Input	Group	Specifies if the master sends its position to one slave or a group of slaves
	SlaveID	The axis ID of the slave or group ID of group of slaves
	ReferenceBase	Specifies if the master sends its load position or its position reference
	Enable	Enable/disables the master in electronic gearing
	SetSlavePos	Specify if the master is initializing the slave(s)
	MoveMoment	Defines the moment when the settings are activated
Output	return	TRUE if no error, FALSE if error

Description: The function programs the active axis as master in electronic gearing. Once at each slow loop sampling time interval, the master sends either its load position APOS (**ReferenceBase = FROM_MEASURE**) or its position reference TPOS (**ReferenceBase = FROM_REFERENCE**) to the axis or the group of axes specified in the parameter **SlaveID**.

Remark: The ReferenceBase = FROM_MEASURE option is not valid if the master operates in open loop. It is meaningless if the master drive has no position sensor.

The **SlaveID** is interpreted either as the Axis ID of one slave (**Group = FALSE**) or the value of a Group ID i.e. the group of slaves to which the master should send its data (**Group = TRUE**).

The master operation is enabled with **Enable = TRUE** and is disabled when **Enable = FALSE**. In both cases, these operations have no effect on the motion executed by the master.

If the master activation is done AFTER the slaves are set in electronic gearing mode, set **SetSlavePos = TRUE** to determine the master to send an initialization message to the slaves.

The commands are executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_SetGearingSlave, TS_SendSynchronization

Associated examples: Ex07_MultiAxes

3.3.6.18 TS_SetGearingSlave

Prototype:

BOOL TML_EXPORT TS_SetGearingSlave(SHORT Denominator, SHORT Numerator, SHORT ReferenceBase, SHORT EnableSlave, DOUBLE LimitVariation, SHORT MoveMoment);

Arguments:

	Name	Description
Input	Denominator	Gear ratio denominator (always positive)
	Numerator	Gear ratio numerator (positive or negative)
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
	EnableSlave	Enables the electronic gearing slave mode
	EnableSuperposition	Enables/disables motion superposition
	LimitVariation	Acceleration limit when the slave is coupling
	MoveMoment	Defines the moment when the settings are activated
Output	return	TRUE if no error, FALSE if error

Description: The function programs the active axis 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 the result of the division **Numerator** / **Denominator**. **Numerator** is a signed integer, while the **Denominator** is unsigned integer. The **Numerator** sign indicates the direction of movement: positive – same as the master, negative – reversed to the master. **Numerator** and **Denominator** 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 (**EnableSlave = SLAVE_COMMUNICATION_CHANNEL**), from a drive/motor set as master with function block TS_SetGearingMaster
2. Via an external digital reference of type pulse & direction or quadrature encoder (**EnableSlave = SLAVE_2ND_ENCODER**)

*Remark: Set **EnableSlave = SLAVE_NONE** if you want to program the motion mode parameters without enabling it.*

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 TS_SetLongVariable to change its value by writing the desired value in the TML variable APOS2.

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. This feature is activated when the parameter **LimitVariation** has a positive value and disabled when its value is zero.

Set **ReferenceBase** = **FROM_REFERENCE** if you want the reference generator to compute the slave position starting from the actual values of the position and speed reference. Set **ReferenceBase** = **FROM_MEASURE** if you want the reference generator to compute the slave position starting from the actual values of the load/motor position and speed. When this option is used, at the beginning of each new motion profile, the position and speed reference are updated with the actual values of the load/motor position and speed.

Remarks:

1. *The function 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. *Use function block **TS_SetGearingMaster** to program a drive/motor as master in electronic gearing*
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 gearing*

Related functions: TS_SetGearingMaster, TS_SetMasterResolution

Associated examples: Ex07_MultiAxes

3.3.6.19 TS_SetCammingMaster

Prototype:

BOOL TML_EXPORT TS_SetCammingMaster(BOOL Group, BYTE SlaveID, SHORT ReferenceBase, BOOL Enable, SHORT MoveMoment);

Arguments:

	Name	Description
Input	Group	Specifies if the master sends its position to one slave or a group of slaves
	SlaveID	The axis ID of the slave or group ID of group of slaves
	ReferenceBase	Specifies if the master sends its load position or its position reference
	Enable	Enable/disables the master in electronic camming
	MoveMoment	Defines the moment when the settings are activated
Output	return	TRUE if no error, FALSE if error

Description: The function programs the active axis as master in electronic camming. Once at each slow loop sampling time interval, the master sends either its load position APOS (**ReferenceBase = FROM_MEASURE**) or its position reference TPOS (**ReferenceBase = FROM_REFERENCE**) to the axis or the group of axes specified in the parameter **SlaveID**.

Remark: The ReferenceBase = FROM_MEASURE option is not valid if the master operates in open loop. It is meaningless if the master drive has no position sensor.

The **SlaveID** is interpreted either as the Axis ID of one slave (**Group = FALSE**) or the value of a Group ID i.e. the group of slaves to which the master should send its data (**Group = TRUE**).

The master operation is enabled with **Enable = TRUE** and is disabled when **Enable = FALSE**. In both cases, these operations have no effect on the motion executed by the master.

The commands are executed:

- Immediately when **MoveMoment** = **UPDATE_IMMEDIATE**
- When a programmed event occurs if **MoveMoment** = **UPDATE_ON_EVENT**
- If you select **MoveMoment** = **UPDATE_NONE**, the movement is parameterized, but does not execute. You'll need to issue an update command to determine the execution of the movement. Use the **TS_UpdateImmediate** or the **TS_UpdateOnEvent** functions in order to activate the movement.

Related functions: TS_CamDownload, TS_CamInitialization TS_SetCammingSlaveRelative, TS_SetCammingSlaveAbsolute, TS_SendSynchronization

Associated examples: Ex07_MultiAxes

3.3.6.20 TS_SetCammingSlaveRelative

Prototype:

BOOL TML_EXPORT TS_SetCammingSlaveRelative(WORD RunAddress, SHORT ReferenceBase, SHORT EnableSlave, SHORT MoveMoment, LONG OffsetFromMaster, DOUBLE MultInputFactor, DOUBLE MultOutputFactor);

Arguments:

	Name	Description
Input	RunAddress	Drive/motor RAM address where the cam table is copied with function TS_CamInitialization
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
	EnableSlave	Enable the electronic camming slave mode
	MoveMoment	Defines the moment when the settings are activated
	OffsetFromMaster	Cam table offset expressed in TML position units
	MultInputFactor	CAM table input scaling factor
	MultOutputFactor	CAM table output scaling factor
Output	return	TRUE if no error, FALSE if error

Description: The function block programs the active axis to operate as slave in electronic camming relative mode. 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. In electronic camming relative mode the output of the cam table is added to the slave actual position.

The cam tables are previously stored in drive/motor non-volatile memory with function **TS_CamDownload**. After download, previously starting the camming slave, you have to initialize the cam table, i.e. to copy it from non-volatile memory to RAM memory. Use function **TS_CamInitialization** to initialize a cam table. The active cam table is selected through parameter **RunAddress**. The **RunAddress** must contain the drive/motor RAM address where the cam table was copied.

The slave can get the master position in two ways:

1. Via a communication channel (**EnableSlave = SLAVE_COMMUNICATION_CHANNEL**), from a drive/motor set as master with function block TS_SetGearingMaster
2. Via an external digital reference of type pulse & direction or quadrature encoder (**EnableSlave = SLAVE_2ND_ENCODER**)

Remark:

1. *Set **EnableSlave = SLAVE_NONE** if you want to program the motion mode parameters without enabling it.*
2. *Use function block **TS_SetCammingMaster** to program a drive/motor as master in electronic camming. 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*

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_SetLongVariable to change its value by writing the desired value in the TML variable APOS2.

With parameter **OffsetFromMaster** 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.

You can compress/extend the cam table input. Set the parameter **MultInputFactor** with 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 **MultOutputFactor** 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.

Remark: *If the OffsetFromMaster, MultInputFactor and/or MultOutputFactor are set to zero the drive/motor will use the value previously set for the parameter or the default value. With this option the TML code generated by this function is reduced.*

Related functions: TS_CamDownload, TS_CamInitialization, TS_SetCammingSlaveAbsolute, TS_SetCammingMaster, TS_SetMasterResolution,

Associated examples: Ex07_MultiAxes

3.3.6.21 TS_SetCammingSlaveAbsolute

Prototype:

BOOL TML_EXPORT TS_SetCammingSlaveAbsolute(WORD RunAddress, DOUBLE LimitVariation, SHORT ReferenceBase, SHORT EnableSlave, SHORT MoveMoment, LONG OffsetFromMaster, DOUBLE MultInputFactor, DOUBLE MultOutputFactor);

Arguments:

	Name	Description
Input	RunAddress	Drive/motor RAM address where the cam table is copied with function TS_CamInitialization
	LimitVariation	Slave speed limit value expressed in TML speed units
	ReferenceBase	Specifies how the motion reference is computed: from actual values of position and speed reference or from actual values of load/motor position and speed
	EnableSlave	Enable the electronic camming slave mode
	MoveMoment	Defines the moment when the settings are activated
	OffsetFromMaster	Cam table offset expressed in TML position units
	MultInputFactor	CAM table input scaling factor
	MultOutputFactor	CAM table output scaling factor
Output	return	TRUE if no error, FALSE if error

Description: The function block programs the active axis to operate as slave in electronic camming absolute mode. 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. In electronic camming absolute mode the output of the cam table represents the position to reach.

The electronic camming absolute mode may generate abrupt variations on the slave position reference, mainly at entry in the camming mode. Set parameter **LimitVariation** to limit the speed of the slave during travel towards the position to reach. The limitation is disabled if the **LimitVariation** is set to zero.

The cam tables are previously stored in drive/motor non-volatile memory with function TS_CamDownload. After download, previously starting the camming slave, you have to initialize the cam table, i.e. to copy it from non-volatile memory to RAM memory. Use function TS_CamInitialization to initialize a cam table. The active cam table is selected through parameter **RunAddress**. The **RunAddress** must contain the drive/motor RAM address where the cam table was copied.

The slave can get the master position in two ways:

1. Via a communication channel (**EnableSlave = SLAVE_COMMUNICATION_CHANNEL**), from a drive/motor set as master with function block TS_SetGearingMaster
2. Via an external digital reference of type pulse & direction or quadrature encoder (**EnableSlave = SLAVE_2ND_ENCODER**)

Remark:

1. *Set **EnableSlave = SLAVE_NONE** if you want to program the motion mode parameters without enabling it.*
2. *Use function block **TS_SetCammingMaster** to program a drive/motor as master in electronic camming. 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*

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_SetLongVariable to change its value by writing the desired value in the TML variable APOS2.

Set the parameter **OffsetFromMaster** to 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.

You can compress/extend the cam table input. Set the parameter **MultInputFactor** with 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 **MultOutputFactor** 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.

Remark: *If the OffsetFromMaster, MultInputFactor and/or MultOutputFactor are set to zero the drive/motor will use the value previously set for the parameter or the default value. With this option the TML code generated by this function is reduced.*

Related functions: TS_CamDownload, TS_CamInitialization, TS_SetCammingSlaveRelative, TS_SetCammingMaster, TS_SetMasterResolution,

Associated examples: –

3.3.6.22 TS_CamDownload

Prototype:

BOOL TML_EXPORT TS_CamDownload(LPCSTR pszCamFile, WORD LoadAddress, WORD RunAddress, WORD& wNextLoadAddr, WORD& wNexLoadAddr);

Arguments:

	Name	Description
Input	pszCamFile	The name of the file containing the cam table description
	LoadAddress	The non-volatile memory (EEPROM) address where the cam table is downloaded
	RunAddress	The RAM address where the cam table is copied at initialization
Output	wNextLoadAddr	Next available EEPROM address from where a cam table can be downloaded
	wNextRunAddr	Next available RAM address where a cam table can be copied
	return	TRUE if no error, FALSE if error

Description: The function downloads a cam table in the drive/motor non-volatile memory (EEPROM) starting with address **LoadAddress**. The **RunAddress** parameter is required to compute the **wNextRunAddr**. The function returns the next valid memory addresses for cam tables trough output parameters **wNextLoadAddr** respectively **wNextRunAddr**. If the values returned by the function are 0 then there is no memory available.

The **LoadAddress** and **RunAddress** for the **first** cam table downloaded are computed by **EasyMotion Studio** and displayed in the dialogue **Memory Settings**. To open the dialogue **Memory Settings** select the appropriate TML application and in **Application General Information** press the button **Memory Settings**. For the next cam tables, if available, the **LoadAddress** and **RunAddress** are the values returned by the previous call of function TS_CamDownload (parameters wNextLoadAddr and wNextRunAddr).

The cam table description is read from the file **pszCamFile**. The file is generated from **EasyMotion Studio** and has the extension ***.cam**.

Steps to follow when using cam tables:

1. Create or import a cam table in **EasyMotion Studio**. The cam table is saved by EasyMotion Studio as a *.cam file in the application directory.
2. Download the cam table in the drive/motor non-volatile memory with **TS_CamDownload**
3. Initialize the cam table with **TS_CamInitialization** function
4. Program the drive/motor to operate as slave in electronic camming mode with **TS_SetCammingSlaveAbsolute** or **TS_SetCammingSlaveRelative**. Select the cam table used with the parameter RunAddress.

Related functions: TS_SetCammingSlaveRelative, TS_SetCammingSlaveAbsolute, TS_CamInitialization

Associated examples: Ex07_Multiaxes

3.3.6.23 TS_CamInitialization

Prototype:

BOOL TML_EXPORT TS_CamInitialization(WORD LoadAddress, WORD RunAddress);

Arguments:

	Name	Description
Input	LoadAddress	Non-volatile memory (EEPROM) address where the cam table is downloaded
	RunAddress	RAM address where the cam table is copied at run time
Output	return	TRUE if no error, FALSE if error

Description: The function copies a cam table from drive/motor non-volatile memory in the RAM memory at address **RunAddress**. The cam table was previously downloaded with function **TS_CamDownload** at non-volatile memory address **LoadAddress**.

The function must be called for each cam table used by the application.

Related functions: TS_SetCammingSlaveRelative, TS_SetCammingSlaveAbsolute, TS_CamDownload

Associated examples: Ex07_MultiAxes

3.3.6.24 TS_SetMasterResolution

Prototype:

BOOL TML_EXPORT TS_SetMasterResolution(LONG MasterResolution);

Arguments:

	Name	Description
Input	MasterResolution	Number of encoder counts per one revolution of the master position sensor.
Output	return	TRUE if no error, FALSE if error

Description: The function sets the TML parameter **MASTERRES** with the value **MasterResolution**.

The master resolution is needed by the electronic gearing or camming slaves to compute correctly the master position and speed (i.e. the position increment). If master position is not cyclic (i.e. the resolution is equal with the whole 32-bit range of position), set master resolution to 0x80000001.

*Remark: Call function **TS_SetMasterResolution** before activating the electronic gearing or camming slave mode with function **TS_SetGearingSlave** respectively **TS_SetCammingSlaveAbsolute/Relative**.*

Related functions: TS_SetGearingSlave, TS_SetCammingSlaveAbsolute, TS_SetCammingSlaveRelative

Associated examples: Ex07_MultiAxes

3.3.6.25 TS_SendSynchronization

Prototype:

BOOL TML_EXPORT TS_SendSynchronization(LONG Period);

Arguments:

	Name	Description
Input	Period	Time period between two synchronization messages. It is expressed in TML time units
Output	return	TRUE if no error, FALSE if error

Description: The function enables/disables the synchronization procedure between axes. The synchronization process is activated when the parameter **Period** has a non-zero value. The active axis is set as the synchronization master and the other axes become synchronization slaves. To disable the synchronization procedure set the **Period** to zero.

The synchronization process is performed in two steps. First, the master sends a synchronization message to all axes, including to itself. When this message is received, all the axes read their own internal time. Next, the master sends its internal time to all the slaves, which compare it with their own internal time. If there are differences, the slaves correct slightly their sampling periods in order to keep them synchronized with those of the master. As effect, when synchronization procedure is active, the execution of the control loops on the slaves is synchronized with those of the master within a 10µs time interval. Due to this powerful feature, drifts between master and slave axes are eliminated. The **Period** represents the time interval in internal units between the synchronization messages sent by the master. Recommended value is 20ms.

Related functions: TS_SetGearingMaster, TS_SetGearingSlave TS_SetCammingMaster, TS_SetCammingSlave

Associated examples: –

3.3.7 Motor commands

3.3.7.1 TS_Power

Prototype:

BOOL TML_EXPORT TS_Power(BOOL Enable);

Arguments:

	Name	Description
Input	Enable	Enables/disables the power stage of the active axis
Output	return	TRUE if no error; FALSE if error

Description: The function enables/disables the power stage of the active axis. If **Enable = TRUE** the power stage is enabled (executes the TML command **AxisON**). The power stage is disabled (executes the TML command **AxisOFF**) when **Enable = FALSE**.

Related functions: TS_ResetFault, TS_Reset

Associated examples: All

3.3.7.2 TS_UpdateImmediate

Prototype:

BOOL TML_EXPORT TS_UpdateImmediate(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The function updates the motion mode immediately. It allows you to start a motion previously programmed. This can be useful for example if you already defined a motion and you want to start it in a specific context (after testing a condition, event, input port, etc.). The command can also be useful to repeat the last motion that was already defined and eventually executed (as for example a relative move).

Related functions: TS_UpdateOnEvent

Associated examples: Ex08_PVT

3.3.7.3 TS_UpdateOnEvent

Prototype:

BOOL TML_EXPORT TS_UpdateOnEvent(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The function updates the motion mode on next event occurrence. It allows you to start a motion that was previously programmed at the occurrence of the active event. This can be useful for example if you already defined a motion and you want to start it when an event occurs. The command can also be used to repeat the last motion that was already defined and eventually executed (as for example a relative move), when the event will occur.

Related functions: TS_UpdateImmediate

Associated examples:

3.3.7.4   TS_Stop

Prototype:

BOOL TML_EXPORT TS_Stop(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The functions stops the motor with the deceleration rate set in TML parameter CACC. The drive/motor decelerates following a trapezoidal speed profile. If the function is called during the execution of an S-curve profile, the deceleration profile may be chosen between a trapezoidal or an S-curve profile. You can detect when the motor has stopped by setting a motion complete event with function **TS_SetEventOnMotionComplete** and waiting until the event occurs (**WaitEvent** = **TRUE**). When the drive performs torque control the drive is set in torque external mode with current reference = 0.

Remarks:

- *In order to restart after a TS_Stop call you need to set again the motion mode. This operation disables the stop mode and allows the motor to move*
- *When TS_Stop is executed it will automatically stop any TML program execution, to avoid overwriting the command from the TML program*
- *During abrupt stops an important energy may be generated. If the power supply can't absorb the energy generated by the motor, it is necessary to foresee an adequate surge capacitor in parallel with the drive supply to limit the over voltage.*

Related functions: TS_QuickStopDecelerationRate

Associated examples: Ex02_DriveStatus,   Ex04_BasicMove,   Ex05_Homing, Ex06_ExternalReference

3.3.7.5 TS_SetPosition

Prototype:

BOOL TML_EXPORT TS_SetPosition(long PosValue);

Arguments:

	Name	Description
Input	PosValue	The value used to set the position, expressed in TML position units
Output	return	TRUE if no error, FALSE if error

Description: The function sets/changes the referential for position measurement by changing simultaneously the load position (TML variable APOS) and the target position values (TML variable APOS), while keeping the same position error. Future motion commands will then be related to the absolute value, as updated at this point to **PosValue**.

Related functions: –

Associated examples: Ex04_BasicMove, Ex05_Homing, Ex07_MultiAxes, Ex09_Logger,Ex11_IOHandling

3.3.7.6 TS_SetTargetPositionToActual

Prototype:

BOOL TML_EXPORT TS_SetTargetPositionToActual(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The function sets the value of the target position (the position reference) to the value of the actual load position i.e. TPOS = APOS_LD. The command may be used in closed loop systems when the load/motor is still following a hard stop, to reposition the target position to the actual load position.

Remark: The command is automatically done if the next motion mode is set with ReferenceBase = FROM_MEASURE. In this case the target position and speed are both updated with the actual values of the load position and respectively load speed: TPOS = APOS_LD and TSPD = ASPD_LD.

Related functions: –

Associated examples: –

3.3.7.7 TS_SetCurrent

Prototype:

BOOL TML_EXPORT TS_SetCurrent(short CrtValue);

Arguments:

	Name	Description
Input	CrtValue	Value at which the motor current reference is set expressed in TML current units
Output	Return	TRUE if no error, FALSE if error

Description: The function sets the motor run current with **CrtValue**. The run current is used by the drive to control the step motor in open loop.

Remark: The command is valid only for configurations with step motor operating in open loop.

Related functions: –

Associated examples: –

3.3.7.8 TS_QuickStopDecelerationRate

Prototype:

BOOL TML_EXPORT TS_QuickStopDecelerationRate(DOUBLE Deceleration);

Arguments:

	Name	Description
Input	Deceleration	The value written in TML parameter CDEC
Output	return	TRUE if no error, FALSE if error

Description: The function sets on the active axis the TML parameter **CDEC** with the value **Deceleration**. The drive/motor uses the deceleration rate when:

- The function TS_Stop is executed during a positioning set with TS_MoveSCurveRelative/Absolute and option **DecelerationType** = **TRAPEZOIDAL_SPEED_PROFILE**
- Enters in **quick stop mode.** The drive enters in quick stop mode if an error requiring the immediate stop of the motion occurs (like triggering a limit switch or following a command error), the drive/motor enters automatically

Related functions: TS_Stop, TS_MoveSCurveRelative, TS_MoveSCurveAbsolute

Associated examples: Ex05_Homing, Ex11_IOHanding

3.3.8   Events

3.3.8.1   TS_CheckEvent

Prototype:

BOOL TML_EXPORT TS_CheckEvent(BOOL &event);

Arguments:

	Name	Description
Input	–	–
Output	Event	Signal if event occurred
	return	TRUE if no error, FALSE if error

Description: The function checks if the actually active event occurred. If an event was defined using one of the **SetEvent…** functions with **WaitEvent = FALSE** then you can check if the event occurred using the **TS_CheckEvent** function.

This is an interesting alternative to the case when **WaitEvent** parameter was set to TRUE in one of the **SetEvent…** functions. In that case, if the event will not occur, due to some unexpected problems, the program will hang-up in an internal loop of the **SetEvent…** function, waiting for the event to occur.

Thus, in order to avoid such a problem, set the **WaitEvent** parameter to FALSE, in the **SetEvent…** function, and then call the **TS_CheckEvent** function from your application. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

Related functions: all SetEvent… functions

Associated examples: Ex10_EventHandling

3.3.8.2 TS_SetEventOnMotionComplete

Prototype:

BOOL TML_EXPORT TS_SetEventOnMotionComplete(BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	WaitEvent	–
	EnableStop	On motion complete stop the motion
Output	return	TRUE if no error, FALSE if error

Description: The function sets an event when the motion is completed. You can use, for example, this event to start your next move only after the actual move is finalized.

The motion complete condition is set in the following conditions:

- During position control:
    - If UPGRADE.11=1, when the position reference arrives at the position to reach (commanded position) and the position error remains inside a settle band for a preset stabilize time interval. The settle band is set with TML parameter POSOKLIM and the stabilize time with TML parameter TONPOSOK. This is the default condition.
    - If UPGRADE.11=0, when the position reference arrives at the position to reach (commanded position)
- During speed control, when the speed reference arrives at the commanded speed

The motion complete condition is reset when a new motion is started i.e. when the update command – UPD is executed.

Remark:

1. *Use function TS_SetIntVariable to change the settle band and/or the stabilize time.*
2. *In case of steppers controlled open-loop, the motion complete condition for positioning is always set when the position reference arrives at the position to reach independently of the UPGRADE.11 status.*

If the **WaitEvent = TRUE**, the function will continuously test the status of the drive event, and will wait until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnMotionComplete** waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At event occurrence the motion is stopped if the parameter **EnableStop = TRUE**. Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

---

Associated examples: Ex04_BasicMove, Ex05_Homing, Ex07_MultiAxes, Ex09_Logger, Ex11_IOHandling

3.3.8.3 TS_SetEventOnMotorPosition

Prototype:

BOOL TML_EXPORT TS_SetEventOnMotorPosition(SHORT PositionType, LONG Position, BOOL Over, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	PositionType	Specifies the motor position type: absolute or relative
	Position	The position value that triggers the event expressed in TML position units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of motor position. The events can be: when the absolute (**PositionType = ABSOLUTE_POSITION)** or relative (**PositionType = ABSOLUTE_RELATIVE**) motor position is equal or over/under **Position**.

The absolute motor position is the measured position of the motor. The relative position is the load displacement from the beginning of the actual movement. For example if a position profile was started with the absolute load position 50 revolutions, when the absolute load position reaches 60 revolutions, the relative motor position is 10 revolutions.

The condition monitored for the event is set with parameter **Over.** For **Over = TRUE** the event is set when the motor position is equal or over the **Position**. When **Over = FALSE** the event is set if the motor position becomes equal or under **Position**.

If the **WaitEvent = TRUE**, the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnMotorPosition** waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At event occurrence the motion is stopped if the parameter **EnableStop = TRUE**. Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples: Ex10_EventHandling

3.3.8.4   TS_SetEventOnLoadPosition

Prototype:

BOOL TML_EXPORT TS_SetEventOnLoadPosition(LONG Position, BOOL Over, BOOL WaitEvent, SHORT EnableStop);

Arguments:

	Name	Description
Input	PositionType	Specifies the load position type: absolute or relative
	Position	The position value that triggers the event expressed in TML position units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of load position. The events can be: when the absolute (**PositionType = ABSOLUTE_POSITION)** or relative (**PositionType = ABSOLUTE_RELATIVE**) load position is equal or over/under **Position**.

The absolute load position is the measured position of the load. The relative position is the load displacement from the beginning of the actual movement.

The condition monitored for the event is set with parameter **Over.** For **Over = TRUE** the event is set when the load position is equal or over the **Position**. When **Over = FALSE** the event is set if the load position becomes equal or under **Position**.

If the **WaitEvent = TRUE**, the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnLoadPosition** waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At event occurrence the motion is stopped if the parameter **EnableStop = TRUE**. Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples:

3.3.8.5 TS_SetEventOnMotorSpeed

Prototype:

BOOL TML_EXPORT TS_SetEventOnMotorSpeed(DOUBLE Speed, BOOL Over, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	Speed	The speed value that triggers the event expressed in TML speed units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	Return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of motor speed. The events can be: when the motor speed is over (**Over = TRUE**) or under (**Over = FALSE**) the **Speed** parameter.

If the **WaitEvent = TRUE**, the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnMotionComplete** waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At event occurrence the motion is stopped if the parameter **EnableStop = TRUE**. Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples: Ex04_BasicMove, Ex10_EventHandling

3.3.8.6   TS_SetEventOnLoadSpeed

Prototype:

BOOL TML_EXPORT TS_SetEventOnLoadSpeed(DOUBLE Speed, BOOL Over, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	Speed	The speed value that triggers the event expressed in TML speed units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of load speed. The events can be: when the load speed is over (**Over = TRUE**) or under (**Over = FALSE**) the **Speed** parameter.

If the **WaitEvent = TRUE**, the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnLoadSpeed** waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At event occurrence the motion is stopped if the parameter **EnableStop = TRUE**. Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples:

3.3.8.7 TS_SetEventOnTime

Prototype:

BOOL TML_EXPORT TS_SetEventOnTime(WORD Time, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	Time	Time delay expressed in TML time units
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	On event stop the motion
Output	return	TRUE if no error, FALSE if error

Description: The function programs an event after a time period equal to the value of the **Time** parameter.

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnTime** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Remark: The timers start ONLY after the execution of the ENDINIT (end of initialization) command. Therefore you should not set wait events before calling the TS_DriveInitialization function..

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples: Ex04_BasicMove, Ex06_ExternalReference

3.3.8.8   TS_SetEventOnPositionRef

Prototype:

BOOL TML_EXPORT TS_SetEventOnPositionRef(LONG Position,   BOOL Over, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	Position	The position reference value that triggers the event expressed in TML position units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of position reference. Setting this event you can detect when the position reference is over (**Over = TRUE**) or under (**Over = FALSE**) the value of parameter **Position**.

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnPositionRef** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples: Ex07_MultiAxes

3.3.8.9   TS_SetEventOnSpeedRef

Prototype:

BOOL TML_EXPORT TS_SetEventOnSpeedRef(DOUBLE Speed,   BOOL Over, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	Speed	The speed reference value that triggers the event expressed in TML speed units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of speed reference. Setting this event you can detect when the speed reference is over (**Over = TRUE**) or under (**Over = FALSE**) the value of parameter **Speed**.

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnSpeedRef** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples:

3.3.8.10 TS_SetEventOnTorqueRef

Prototype:

BOOL TML_EXPORT TS_SetEventOnTorqueRef(DOUBLE Torque, BOOL Over, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	Torque	The torque reference value that triggers the event expressed in TML current units.
	Over	Specifies the condition tested
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of torque reference. Setting this event you can detect when the torque reference is over (**Over = TRUE**) or under (**Over = FALSE**) the value of parameter **Torque**.

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnTorqueRef** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples:

3.3.8.11 TS_SetEventOnEncoderIndex

Prototype:

BOOL   TML_EXPORT   TS_SetEventOnEncoderIndex(SHORT   IndexType,   SHORT TransitionType, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	IndexType	Specifies the index monitored for transition
	TransitionType	Specifies the input transition monitored
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of drive/motor encoder index inputs. You can monitor the first encoder index (**IndexType = Index_1**) or the second encoder index (**IndexType = Index_2**). The event is trigger by encoder index transition low to high when **TransitionType = TRANSITION_LOW_TO_HIGH** or by the transition high to low when **TransitionType = TRANSITION_ HIGH_TO_LOW.**

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnEncoderIndex** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent, and all other SetEvent… functions

Associated examples:

3.3.8.12 TS_SetEventOnLimitSwitch

Prototype:

BOOL TML_EXPORT TS_SetEventOnLimitSwitch(SHORT LSWType, SHORT TransitionType, BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	LSWType	Specifies the limit switch monitored for transition
	TransitionType	Specifies the input transition monitored
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of drive/motor limit switch inputs. The event is set:

- when a transition occurs on limit switch negative if parameter **LSWType = LSW_NEGATIVE**
- when a transition occurs on limit switch negative if parameter **LSWType = LSW_POSITIVE**

You can monitor the limit switch transition low to high when **TransitionType = TRANSITION_LOW_TO_HIGH** or the transition high to low when **TransitionType = TRANSITION_ HIGH_TO_LOW.**

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnLimitSwitch** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent and all other SetEvent… functions

Associated examples: Ex05_Homing

3.3.8.13 TS_SetEventOnDigitalInput

Prototype:

BOOL TML_EXPORT TS_SetEventOnDigitalInput(BYTE InputPort, SHORT IOState BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
Input	InputPort	Specifies the digital input monitored
	IOState	The input state that trigger the event
	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of drive/motor general purpose digital inputs. The event is set when a transition occurs on digital input **InputPort**.

You can monitor when the digital input goes high (**IOState = IO_HIGH)** or the digital input goes low (**IOState = IO_LOW**).

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnDigitalInput** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent and all other SetEvent… functions

Associated examples:

3.3.8.14 TS_SetEventOnHomeInput

Prototype:

BOOL TML_EXPORT TS_SetEventOnHomeInput(SHORT IOState BOOL WaitEvent, BOOL EnableStop);

Arguments:

	Name	Description
	IOState	Input port state (High/low)
Input	WaitEvent	Specifies if the function waits the event occurrence
	EnableStop	Stop the motion when at event occurrence
Output	return	TRUE if no error, FALSE if error

Description: It allows you to program an event function of drive/motor general purpose digital input assigned as home input. The home input is specific for each product and based on the setup data. The event is set when a transition occurs on home input.

You can monitor when the home input goes high (**IOState = IO_HIGH**) or the home input goes low (**IOState = IO_LOW**).

If the parameter **WaitEvent = TRUE** the function tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected problems. In such a case, the program hangs-up in an internal loop of the **TS_SetEventOnHomeInput** function, waiting for the event to occur.

If the parameter **WaitEvent = FALSE** you can check if the event occurred using the **TS_CheckEvent** function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

At the event occurrence the motion is stopped if the parameter **EnableStop = TRUE.** Set **EnableStop = FALSE** if you do not want to stop the motion at event occurrence.

Related functions: TS_CheckEvent and all other SetEvent… functions

Associated examples:

3.3.9 TML jumps and function calls

3.3.9.1 TS_GOTO

Prototype:

BOOL TML_EXPORT TS_GOTO(WORD address);

Arguments:

	Name	Description
Input	address	The memory address where the jump is made
Output	return	TRUE if no error, FALSE if error

Description: The function commands the active axis to execute the TML code beginning from the **address** until TML instruction **END** is encountered. The TML code can be stored in the drive/motor non-volatile memory (EEPROM) or in the TML program memory.

Prior calling the **TS_GOTO** function you have to:

- Create a TML sequence using **EasyMotion Studio**
- Create the executable file (***.out**) with the menu command **Application | Motion | Build**
- Download the TML code in the drive/motor memory with EasyMotion Studio or function **TS_DownloadProgram**
- Make sure that a valid instruction is found at **address**. Otherwise, unpredictable effects can occur, which can affect to correct operation of the drive/motor.

Remark:

1. *For more details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio help.*
2. *During the execution of a local TML program on the drive, any TML command sent on-line from the PC is treated with higher priority, and will be executed before executing the local TML code.*

Related functions: TS_DownloadProgram, TS_GOTO_Label, TS_CALL, TS_CALL_Label

Associated examples: Ex08_PVT, Ex12_DistributedTasks

3.3.9.2 TS_GOTO_Label

Prototype:

BOOL TML_EXPORT TS_GOTO_Label(LPCSTR pszLabel);

Arguments:

	Name	Description
Input	pszLabel	TML program label where the jump is made
Output	return	TRUE if no error, FALSE if error

Description: The function commands the active axis to execute the TML code beginning from label **pszLabel** until TML instruction **END** is encountered. The TML code can be stored in the drive/motor non-volatile memory (EEPROM) or in the TML program memory.

The string **pszLabel** must be a valid TML label, defined in EasyMotion Studio prior generating the setup information.

Prior calling the **TS_GOTO_Label** function you have to:

- Create a TML motion sequence using **EasyMotion Studio**. The commands sequence must start with **pszLabel** declaration.
- Create the executable file (***.out**) with the menu command **Application | Motion | Build**
- Download the TML code in the drive/motor memory with EasyMotion Studio or function TS_DownloadProgram
- Generate the setup data (***.t.zip**) for TML_lib using the menu command **Application | Export to TML_lib…** to include the new **pszLabel**

Remark:

1. *For more details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio help.*
2. *During the execution of a local TML program on the drive, any TML command sent on-line from the PC is treated with higher priority, and will be executed before executing the local TML code.*

Related functions: TS_DownloadProgram, TS_GOTO, TS_CALL, TS_CALL_Label, TS_CancelableCALL, TS_CancelableCALL_Label

Associated examples:

3.3.9.3 TS_CALL

Prototype:

BOOL TML_EXPORT TS_CALL (WORD address);

Arguments:

	Name	Description
Input	address	The memory address where the jump is made
Output	return	TRUE if no error, FALSE if error

Description: The function commands the active axis to execute the TML function stored at **address**. The TML functions can be stored in the drive/motor non-volatile memory or in the TML program memory. The function execution ends when the TML instruction **RET** is encountered.

Prior calling the **TS_CALL** function you have to:

- Create at least one TML function using **EasyMotion Studio**
- Select, in the **Memory Setting** dialogue, from where you want to run the TML program: TML program or non-volatile memory.
- Create the executable file (***.out**) with the menu command **Application | Motion | Build**
- In the **Command Interpreter** type the command **?Function_name** to retrieve the memory address where the **Function_name** is stored. Repeat the above procedure for all the functions defined in EasyMotion Studio
- Download the TML code in the drive/motor memory with EasyMotion Studio or function **TS_DownloadProgram**
- Make sure that a valid TML code subroutine begins at **address**. Otherwise, unpredictable effects can occur, which can affect to correct operation of the drive/motor.

Remark:

1. *For more details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio help.*
2. *During the execution of a local TML program on the drive, any TML command sent on-line from the PC is treated with higher priority, and will be executed before executing the local TML code.*

Related functions: TS_DownloadProgram, TS_CALL_Label, TS_CancelableCALL, TS_CancelableCALL_Label, TS_GOTO, TS_GOTO_label

Associated examples:

3.3.9.4   TS_CALL_Label

Prototype:

BOOL TML_EXPORT TS_CALL_Label (LPCSTR pszFunctionName);

Arguments:

	Name	Description
Input	pszFunctionName	Name of the TML function
Output	return	TRUE if no error, FALSE if error

Description: The function commands the active axis to execute the TML function **pszFunctionName**. The TML functions can be stored in the drive/motor non-volatile memory or in the TML program memory. The function execution ends when the TML instruction **RET** is encountered.

The string **pszFunctionName** must be a valid TML function name, defined in EasyMotion Studio prior generating the setup information.

Prior calling the **TS_CALL_Label** function you have to:

- Create a TML function having the name **pszFunctionName** using **EasyMotion Studio**
- Select, in the **Memory Setting** dialogue, from where you want to run the TML program: TML program or non-volatile memory (EEPROM).
- Create the executable file (***.out**) with the menu command **Application | Motion | Build**
- Generate the setup data (***.t.zip**) for TML_lib using the menu command **Application | Export to TML_lib…** to include the new **pszFunctionName**
- Download the TML code in the drive/motor memory with EasyMotion Studio or function **TS_DownloadProgram**

Remark:

1. *For more details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio help.*
2. *During the execution of a local TML program on the drive, any TML command sent on-line from the PC is treated with higher priority, and will be executed before executing the local TML code.*

Related functions: TS_DownloadProgram, TS_CALL, TS_GOTO, TS_GOTO_Label, TS_CancelableCALL, TS_CancelableCALL_Label

Associated examples:

3.3.9.5 TS_CancelableCALL

Prototype:

BOOL TML_EXPORT TS_CancelableCALL(WORD address);

Arguments:

	Name	Description
Input	address	The TML program memory address from where the TML function is stored
Output	return	TRUE if no error, FALSE if error

Description: The function commands the active axis to execute the TML function stored at **address**. Use this command if the exit from the called TML function depends on conditions that may not be reached. In this case, using function **TS_Abort** you can terminate the function execution and return to the next instruction after the call. The TML functions can be stored in the drive/motor non-volatile memory or in the TML program memory. The execution of a function called with TS_CancelableCALL is signaled with bit SRL.8, when the function execution ends the bit is reset.

Prior calling the **TS_CancelableCALL** function you have to:

- Create at least one TML function using **EasyMotion Studio**
- Select, in the **Memory Setting** dialogue, from where you want to run the TML program: TML program or non-volatile memory.
- Create the executable file (***.out**) with the menu command **Application | Motion | Build**
- Download the TML code in the drive/motor memory with EasyMotion Studio or function **TS_DownloadProgram**
- In the **Command Interpreter** type the command **?Function_name** to retrieve the memory address where the **Function_name** is stored. Repeat the procedure above for all the functions defined in EasyMotion Studio.
- Make sure that a valid TML code subroutine begins at **address**. Otherwise, unpredictable effects can occur, which can affect to correct operation of the drive/motor.

Remark:

1. *You can call only one function at a time using the TS_CancelableCALL. Any cancelable call issued during the execution of a function called with TS_CancelableCALL is ignored. This situation is signaled with bit SRL.7.*
2. *For more details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio help.*
3. *During the execution of a local TML program on the drive, any TML command sent on-line from the PC is treated with higher priority, and will be executed before executing the local TML code.*

Related functions: TS_DownloadProgram, TS_CALL, TS_CALL_Label, TS_CancelableCALL_Label

Associated examples:

3.3.9.6   TS_CancelableCALL_Label

Prototype:

BOOL TML_EXPORT TS_CancelableCALL_Label (LPCSTR pszFunctionName);

Arguments:

	Name	Description
Input	pszFunctionName	Name of the TML function
Output	return	TRUE if no error, FALSE if error

Description: The function commands the active axis to execute the TML function stored at **pszFunctionName**. Use this command if the exit from the called TML function depends on conditions that may not be reached. In this case, using function **TS_Abort** you can terminate the TML function execution and return to the next instruction after the call. The execution of a function called with TS_CancelableCALL_Label is signaled with bit SRL.8, when the function execution ends the bit is reset.

Prior calling the **TS_CancelableCALL_Label** function you have to:

- Create a TML function having the name **pszFunctionName** using **EasyMotion Studio**
- Select, in the **Memory Setting** dialogue, from where you want to run the TML program: TML program or non-volatile.
- Create the executable file (***.out**) with the menu command **Application | Motion | Build**
- Generate the setup data (***.t.zip**) for TML_lib using the menu command **Application | Export to TML_lib…** to include the new **pszFunctionName**
- Download the TML code in the drive/motor memory with EasyMotion Studio or function **TS_DownloadProgram**

Remarks:

1. *You can call only one function at a time using the TS_CancelableCALL. Any cancelable call issued during the execution of a function called with TS_CancelableCALL is ignored. This situation is signaled with bit SRL.7.*
2. *For more details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio help.*
3. *During the execution of a local TML program on the drive, any TML command sent on-line from the PC is treated with higher priority, and will be executed before executing the local TML code.*

Related functions: TS_DownloadProgram, TS_CALL, TS_CALL_Label, TS_CancelableCALL

Associated examples: Ex05_Homing, Ex12_DistributedTasks

3.3.9.7 TS_ABORT

Prototype:

BOOL TML_EXPORT TS_Abort(void);

Arguments:

	Name	Description
Input	–	–
Output	return	TRUE if no error, FALSE if error

Description: The function aborts the execution of a TML function launched with a cancelable call. The aborted function is selected with parameter **pszFunctionName**. The functions labels for a setup configuration are listed in the **variables.cfg** file.

*Remark: The TML functions must be created with **EasyMotion Studio** prior generating the setup information.*

 Related functions: TS_DownloadProgram, TS_CancelableCALL, TS_CancelableCALL_Label

Associated examples: Ex05_Homing, Ex12_DistributedTasks

3.3.9.8 TS_DownloadProgram

Prototype:

BOOL TML_EXPORT TS_DownloadProgram(LPCSTR pszOutFile, WORD& wEntryPoint);

Arguments:

	Name	Description
Input	pszOutFile	The name of the out file generated with EasyMotion Studio
Output	wEntryPoint	Start address of downloaded file
	return	TRUE if no error, FALSE if error

Description: The function downloads a COFF formatted file to the drive/motor, and returns the entry point of that file. Parameter **pszOutFile** specifies the name of the object file to be downloaded. If the operation is successful, the function will return the entry point (start address) of the downloaded code in the **wEntryPoint** parameter. You can use this address to launch the execution of the downloaded code, by using it as the input argument of the **TS_GOTO** or **TS_CALL** functions.

The COFF file (***.out**) is generated from EasyMotion Studio with the **Application | Motion | Build** menu command and is saved in the application directory. You can download several such applications in different locations of the drive internal memory, and execute them according to your application status, with the **TS_GOTO** or **TS_CALL** functions.

Related functions: TS_GOTO, TS_CALL

Associated examples: Ex12_DistributedTasks

3.3.9.9 TS_DownloadSwFile

Prototype:

BOOL TML_EXPORT TS_DownloadSwFile(LPCSTR pszSwFile);

Arguments:

	Name	Description
Input	pszSwFile	The path to the SW file generated with EasyMotion Studio
Output	return	TRUE if no error, FALSE if error

Description: The function downloads a software file (*.sw) to the non-volatile memory of drive/motor.

The software file (***.sw**) contains the TML program and/or setup table and is generated from EasyMotion Studio with the **Application | Create EEPROM Programmer File** menu command. You can download several TML programs in different locations of the drive internal memory, and execute them according to your application structure, with the **TS_GOTO** or **TS_CALL** functions.

Remark: If a setup table is downloaded through a software file, it will become active after drive reset.

Related functions: TS_GOTO, TS_CALL

Associated examples: –

3.3.10 IO handling

3.3.10.1 TS_SetupInput

Prototype: BOOL TML_EXPORT TS_SetupInput(BYTE nIO);

Arguments:

	Name	Description
Input	nIO	Port number to be set as input
Output	return	TRUE if no error, FALSE if error

Description: The function sets the I/O port with number **nIO** of the drive/motor as an input port.

Use the function only if the input selected may also be used as an output. **Check the drive/motor user manual to find what inputs are available.** Do this operation only once, first time when you use the input. If the drive/motor has the inputs separated from the outputs (i.e. none of the input line can be used as output) you don't have to use the function.

Remark: Depending on the firmware version programmed on the drive/motor, FAxx or FBxx, the digital inputs and outputs are numbered as follows:

- *from #0 to #39 for firmware **FAxx**[1]. The list is unordered, for example, a product with 4 inputs and 4 outputs can use the inputs: #36, #37, #38 and #39 and the outputs #28, #29, #30 and #31.*
- *From 0 to 15 for firmware version **FBxx**[2]. The list is ordered, for example, a product with 5 inputs and 3 outputs can use the inputs: 0, 1, 2, 3 and 4 and the outputs 0, 1, and 2.*

Each intelligent drive/motor has a specific number of inputs and outputs, therefore only a part of the maximum number of I/Os is used.

Related functions: TS_GetInput, TS_SetupOutput, TS_SetOutput

Associated examples: Ex06_ExternalReference, Ex11_IOHandling, Ex12_DistributedTasks

---

[1] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)

[2] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen

3.3.10.2 TS_GetInput

Prototype:

BOOL TML_EXPORT TS_GetInput(BYTE nIO, BYTE& InValue);

Arguments:

	Name	Description
Input	nIO	Input port number read
	InValue	Pointer to the variable where the port status is stored
Output	return	TRUE if no error, FALSE if error

Description: The function returns the status of digital input port **nIO**. When the function is executed, the variable **InValue,** where the input line status is saved, becomes:

- Zero if the input line was low
- Non-zero if the input line was high

If the IO port selected can be used as input or an output then prior calling **TS_GetInput** you need to call **TS_SetupInput** and configure IO port as input. **Check the drive/motor user manual to find what inputs are available.**

Remark: Depending on the firmware version programmed on the drive/motor, FAxx or FBxx, the digital inputs and outputs are numbered as follows:

- *From #0 to #39 for firmware **FAxx**[1]. The list is unordered, for example, a product with 4 inputs and 4 outputs can use the inputs: #36, #37, #38 and #39 and the outputs #28, #29, #30 and #31.*
- *From 0 to 15 for firmware version **FBxx**[2]. The list is ordered, for example, a product with 5 inputs and 3 outputs can use the inputs: 0, 1, 2, 3 and 4 and the outputs 0, 1, and 2.*

Each intelligent drive/motor has a specific number of inputs and outputs, therefore only a part of the maximum number of I/Os is used.

Related functions: TS_SetupInput, TS_SetupOutput, TS_SetOutput

Associated examples: Ex06_ExternalReference, Ex11_IOHandling, Ex12_DistributedTasks

---

[1] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)
[2] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen

3.3.10.3 TS_SetupOutput

Prototype:

BOOL TML_EXPORT TS_SetupOutput(BYTE nIO);

Arguments:

	Name	Description
Input	nIO	Port number to be set as output
Output	return	TRUE if no error, FALSE if error

Description: The function configures the digital I/O port with number **nIO** of the drive/motor as an output port.

Use the function only if the output selected may also be used as an input. **Check the drive/motor user manual to find what outputs are available.** Do this operation only once, first time when you use the output. If the drive/motor has the outputs separated from the inputs (i.e. none of the output line can be used as an input) you don't have to use the function.

Remark: Depending on the firmware version programmed on the drive/motor, FAxx or FBxx, the digital inputs and outputs are numbered as follows:

- *from #0 to #39 for firmware **FAxx**[1]. The list is unordered, for example, a product with 4 inputs and 4 outputs can use the inputs: #36, #37, #38 and #39 and the outputs #28, #29, #30 and #31*
- *From 0 to 15 for firmware version **FBxx**[2]. The list is ordered, for example, a product with 5 inputs and 3 outputs can use the inputs: 0, 1, 2, 3 and 4 and the outputs 0, 1, and 2.*

Each intelligent drive/motor has a specific number of inputs and outputs, therefore only a part of the maximum number of I/Os is used.

Related functions: TS_GetInput, TS_SetupOutput, TS_SetOutput

Associated examples:

---

[1] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)
[2] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen

3.3.10.4 TS_SetOutput

Prototype:

BOOL TML_EXPORT TS_SetOutput(BYTE nIO, BYTE OutValue);

Arguments:

	Name	Description
Input	nIO	Output port number to be written
	OutValue	Output status value to be set
Output	return	TRUE if no error, FALSE if error

Description: The function set/resets the status of digital output port **nIO** of the drive/motor.

The port status **IO_LOW** or **IO_HIGH** is set corresponding to the value of the **OutValue** parameter.

If the IO port selected may also be used as input or an output then prior calling **TS_SetOutput** you need to call **TS_SetupOutput** and configure IO port as output.

Remark: Depending on the firmware version programmed on the drive/motor, FAxx or FBxx, the digital inputs and outputs are numbered as follows:

- *from #0 to #39 for firmware **FAxx**[1]. The list is unordered, for example, a product with 4 inputs and 4 outputs can use the inputs: #36, #37, #38 and #39 and the outputs #28, #29, #30 and #31*
- *From 0 to 15 for firmware version **FBxx**[2]. The list is ordered, for example, a product with 5 inputs and 3 outputs can use the inputs: 0, 1, 2, 3 and 4 and the outputs 0, 1 and 2.*

Each intelligent drive/motor has a specific number of inputs and outputs, therefore only a part of the maximum number of I/Os is used.

Related functions: TS_SetupOutput, TS_SetupInput, TS_GetInput

Associated examples: Ex11_IOHandling

---

[1] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)
[2] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen

3.3.10.5 TS_GetHomeInput

Prototype:

BOOL TML_EXPORT TS_GetHomeInput(BYTE& InValue);

Arguments:

	Name	Description
Input	InValue	Pointer to the variable where the port status is stored
Output	return	TRUE if no error, FALSE if error

Description: The function returns the status of the general purpose digital input assigned as home input. **Check the drive/motor user manual to find the IO configuration.**

When the function is executed, the variable **InValue** where the input line status is saved becomes:

- Zero if the input line was low
- Non-zero if the input line was high

If the input port may also be used as output then prior calling **TS_GetHomeInput** you need to call **TS_SetupInput** and configure it as input.

Related functions: TS_SetupInput, TS_GetInput

Associated examples: Ex07_MultipleAxes

3.3.10.6 TS_GetMultipleInputs

Prototype:

BOOL TML_EXPORT TS_GetMultipleInputs(PCSTR pszVarName, SHORT& Status);

Arguments:

	Name	Description
Input	pszVarName	TML variable where the inputs status is saved
Output	Status	Pointer to variable where the value of pszVarName is stored
	Return	TRUE if no error, FALSE if error

Description: The function reads simultaneously the status of more inputs and save their status in TML variable **pszVarName** on the drive/motor. The value of **pszVarName** is then uploaded from the drive and stored in **Status** variable.

For drives/motors programmed with firmware version **FAxx**[1] the digital inputs read are:

- Enable input – saved in bit 15 of **pszVarName**
- Limit switch input for negative direction (LSN) – saved in bit 14 of **pszVarName**
- Limit switch input for positive direction (LSP) – saved in bit 13 of **pszVarName**
- General-purpose inputs #39, #38, #37 and #36 – saved in bits 3, 2, 1 and 0 of **pszVarName**

If the drive/motor is programmed with firmware version **FBxx**[2] then the function reads all the input lines available of the drive/motor. The digital inputs are numbered from 0 to 15. The input's number represents also the position of the corresponding bit from the **pszVarName**, i.e. input number **x** has associated bit **x** from the **pszVarName**.

The **Status** bits corresponding to these inputs are set as follows: 0 if the input is low and 1 if the input is high. The other bits of the variable are set to 0.

Remark: If one of these inputs is inverted inside the drive/motor, the corresponding bit from the variable is inverted too. Hence, these bits always show the inputs status at connectors level (0 if input is low and 1 if input is high) even when the inputs are inverted.

The variable **pszVarName** is of type integer and must be defined with EasyMotion Studio before generating the setup data for TML_lib.

Related functions: TS_SetInput, TS_GetInput

Associated examples: Ex11_IOHandling

---

[1] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)
[2] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen

3.3.10.7 TS_SetMultipleOutputs

Prototype:

BOOL TML_EXPORT TS_SetMultipleOutputs(LPCSTR pszVarName, SHORT &Status);

Arguments:

	Name	Description
Input	pszVarName	Intermediary TML variable necessary to store the outputs status to be set on the drive/motor
	Status	Parameter containing the outputs status to be set
Output	return	TRUE if no error, FALSE if error

Description: The function sets simultaneously more digital outputs of the drive/motor using the value of parameter **Status**. Its value is transferred and stored on the drive in **pszVarName** TML variable and from there is used to set the outputs.

*Remark: The function is designed for drives/motors programmed with firmware version **FAxx**[1]. For drives/motors programmed with firmware version **FBxx**[2] use the TS_SetMultipleOutputs2 function.*

The digital outputs are:

- Ready output – set by bit 15 of **pszVarName**
- Error output – set by bit 14 of **pszVarName**
- General-purpose outputs: #31, #30, #29, #28 – set by bits 3, 2, 1, and 0 of **pszVarName**

The outputs are set as follows: low if the corresponding bit in the variable is 0 and high if the corresponding bit in the variable is 1. The other bits of the variable are not used.

Remarks:

1. *If one of these outputs is inverted inside the drive/motor, its command is inverted. Hence, the outputs are always set at connectors level according with the bits values (low if bit is 0 and high if bit is 1) even when the outputs are inverted.*
2. *The variable pszVarName must be declared with EasyMotion Studio prior generating the setup information*

*CAUTION: Do not use **TS_SetMultipleOututs** if any of the 6 outputs mentioned is not on the list of available outputs of your drive/motor. There are products that use some of these outputs internally for other purposes. Attempting to change these lines status may harm your product.*

Related functions: TS_SetupOutput, TS_SetOutput

Associated examples: Ex11_IOHandling

---

[1] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)
[2] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen

3.3.10.8 TS_SetMultipleOutputs2

Prototype:

BOOL TML_EXPORT TS_SetMultipleOutputs2(SHORT SelectedPorts, SHORT &Status);

Arguments:

	Name	Description
Input	SelectedPorts	Mask for selecting the outputs controlled. Each bit of the parameter represents an output port.
	Status	Parameter containing the outputs status to be set
Output	Return	TRUE if no error, FALSE if error

Description: The function sets simultaneously the digital outputs selected with the **SelectedPorts** mask using the value of the **Status** parameter.

*Remark: The function is designed for drives/motors programmed with firmware version **FBxx**[1].* *For drives/motors programmed with firmware version **FAxx**[2] use the TS_SetMultipleOutputs function.*

The digital outputs are numbered from 0 to 15 and they form an ordered list, for example, a product with 3 outputs will have 0, 1 and 2. The input's number represents also the position of the corresponding bit from the **SelectedPorts** mask, i.e. input number **x** has associated bit **x** from the **SelectedPorts**.

The outputs are set as follows:

- **low** if the corresponding bit from the **SelectedPorts** is 1 and the corresponding bit from **Status** variable is 0.
- **high** if it's the corresponding bit from **SelectedPorts** is 1 and the corresponding bit from SelectedPorts is 1.

Remarks: If one of these outputs is inverted inside the drive/motor, its command is inverted. Hence, the outputs are always set at connectors level according with the bits values (low if bit is 0 and high if bit is 1) even when the outputs are inverted.

Related functions: TS_SetupOutput , TS_SetOutput

Associated examples: –

---

[1] Represents the firmware versions F500A or later on Technosoft drives: IDM240 CANopen/IDM640 CANopen, IDS640 CANopen
[2] Represents the firmware versions: F000H, F020H, F005H, F900H or later on Technosoft drives/motors: IDM240/IDM640, IDS240/IDS640, ISCM4805/ISCM8005, IBL2403, IM23x (models IS and MA)

3.3.11 Data transfer

3.3.11.1 TS_SetIntVariable

Prototype:

BOOL TML_EXPORT TS_SetIntVariable(LPCSTR pszName, SHORT value);

Arguments:

	Name	Description
Input	pszName	Parameter name
	value	Parameter value
Output	return	TRUE if no error; FALSE if error

Description: The function writes the **value** in the TML data **pszName** on the active axis. The TML data (parameter, variable or user defined variable) is of type long (16-bit).

Remarks:

1. *The available TML data is configuration dependent and is listed in the variables.cfg file*
2. *The user defined variables are set with EasyMotion Studio prior generating the setup information*

Related functions: TS_GetIntVariable,   TS_SetLongVariable,   TS_GetLongVariable, TS_SetFixedVariable, TS_GetFixedVariable

Associated examples: Ex03_ErrorHandling, Ex06_ExternalReference, Ex12DistributedTasks

3.3.11.2 TS_GetIntVariable

Prototype:

BOOL TML_EXPORT TS_GetIntVariable(LPCSTR pszName, short& value);

Arguments:

	Name	Description
Input	pszName	Name of the TML parameter, variable or used defined variable
	value	Pointer to the variable where the value is stored
Output	return	TRUE if no error, FALSE if error

Description: The function reads the value of TML data **pszName**. The TML data (parameter, variable or user defined variable) is of type integer (16-bit). The value read is saved in the variable pointed by **value**.

Remarks:

1. *The available TML data is configuration dependent and is listed in the **variables.cfg** file.*
2. *The user defined variables are set with EasyMotion Studio prior generating the setup information*

Related functions: TS_SetIntVariable, TS_SetLongVariable, TS_SetFixedVariable, TS_GetLongVariable, TS_GetFixedVariable

Associated examples: Ex03_ErrorHandling, Ex06_ExternalReference

3.3.11.3 TS_SetLongVariable

Prototype:

BOOL TML_EXPORT TS_SetLongVariable(LPCSTR pszName, long value);

Arguments:

	Name	Description
Input	pszName	Name of the parameter
	value	The value to be written
Output	return	TRUE if no error, FALSE if error

Description: The function writes the **value** in the TML data **pszName** on the active axis. The TML data (parameter, variable or user defined variable) is of type long (32-bit).

Remarks:

1. *The available TML data is configuration dependent and is listed in the variables.cfg file*
2. *The user defined variables are set with EasyMotion Studio prior generating the setup information*

Related functions: TS_GetIntVariable, TS_SetIntVariable, TS_GetLongVariable, TS_SetFixedVariable, TS_GetFixedVariable

Associated examples: Ex05_Homing

3.3.11.4 TS_GetLongVariable

Prototype:

BOOL TML_EXPORT TS_GetLongVariable(LPCSTR pszName, long& value);

Arguments:

	Name	Description
Input	pszName	Name of the parameter
	value	Pointer to the variable where the parameter value is stored
Output	return	TRUE if no error, FALSE if error

Description: The function reads the value of TML data **pszName**. The TML data (parameter, variable or user defined variable) is of type long (32-bit). The value read is saved in the variable pointed by **value**.

Remarks:

1. *The available TML data is configuration dependent and is listed in the **variables.cfg** file.*
2. *The user defined variables are set with EasyMotion Studio prior generating the setup information*

Related functions: TS_SetIntVariable, TS_SetLongVariable, TS_SetFixedVariable, TS_GetIntVariable, TS_GetFixedVariable

Associated examples: Ex04_BasicMove, Ex05_Homing, Ex10_EventHandling

3.3.11.5 TS_SetFixedVariable

Prototype:

BOOL TML_EXPORT TS_SetFixedVariable(LPCSTR pszName, DOUBLE value);

Arguments:

	Name	Description
Input	pszName	Name of the parameter
	value	The value to be written
Output	return	TRUE if no error, FALSE if error

Description: The function converts the **value** to type fixed and writes it in the TML data **pszName** on the active axis. The TML data (parameter, variable or user defined variable) is of type fixed (16 bits integer part, 16 bits fractional part).

Remarks:

1. *The available TML data is configuration dependent and is listed in the **variables.cfg** file.*
2. *The user defined variables are set with EasyMotion Studio prior generating the setup information*

Related functions: TS_SetIntVariable, TS_GetIntVariable, TS_SetLongVariable, TS_GetLongVariable, TS_GetFixedVariable

Associated examples: Ex04_BasicMove, Ex05_Homing

3.3.11.6 TS_GetFixedVariable

Prototype:

BOOL TML_EXPORT TS_GetFixedVariable(LPCSTR pszName, double& value);

Arguments:

	Name	Description
Input	pszName	Name of the parameter
	value	Pointer where the parameter value is stored
Output	return	TRUE if no error, FALSE if error

Description: The function reads the value of TML data **pszName** from the active axis. The TML data (parameter, variable or user defined variable) is of type fixed (16 bits integer part, 16 bits fractional part). The value read is converted to double and saved in the variable pointed by **value**.

Remarks:

1. *The available TML data is configuration dependent and is listed in the **variables.cfg** file.*
2. *The user defined variables are set with EasyMotion Studio prior generating the setup information*

Related functions: TS_SetIntVariable, TS_SetLongVariable, TS_SetFixedVariable, TS_GetIntVariable, TS_GetLongVariable

Associated examples:

3.3.11.7 TS_GetVariableAddress

Prototype:

BOOL TML_EXPORT TS_GetVariableAddress(LPCSTR pszName, WORD& value);

Arguments:

	Name	Description
Input	pszName	Name of the parameter
	value	Pointer where the parameter value is stored
Output	return	TRUE if no error, FALSE if error

Description: The function returns the address of **pszName** variable. The variable address is read from the setup data (*.t.zip) generated from EasyMotion Studio.

Related functions: TS_SetIntVariable, TS_SetLongVariable, TS_SetFixedVariable,
TS_GetIntVariable, TS_GetLongVariable, TS_GetFixedVariable,

Associated examples:

3.3.11.8 TS_SetBuffer

Prototype:

BOOL TML_EXPORT TS_SetBuffer(WORD address, WORD* arrayValues, WORD nSize);

Arguments:

	Name	Description
Input	address	Start address where to download the data buffer
	arrayValues	Pointer to the array with data to be downloaded
	nSize	The number of words to download
Output	return	TRUE if no error, FALSE if error

Description: The function downloads a data buffer on the active axis. The parameter **arrayValues** points to the beginning of the array from where the data will be downloaded. The length of the buffer is set with parameter **nSize**. The data is stored on the drive/motor starting with **address**. The **address** can belong to drive/motor non-volatile memory or TML data memory.

Remark: For details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio on line help.

Related functions: TS_GetBuffer

Associated examples:

3.3.11.9 TS_GetBuffer

Prototype:

BOOL TML_EXPORT TS_GetBuffer(WORD address, WORD* arrayValues, WORD nSize);

Arguments:

	Name	Description
Input	address	Start address from where the data will be uploaded
	arrayValues	Pointer to the array where the uploaded data will be stored
	nSize	The number of words to upload
Output	return	TRUE if no error, FALSE if error

Description: The function uploads a data buffer from the active axis. The start address of the buffer is set with parameter **address** and its length is **nSize.** The **address** can belong to drive/motor non-volatile memory or TML data memory. The parameter **arrayValues** points to the beginning of the array where the uploaded data is stored.

Remark: For details about drive/motor memory structure see the "Memory Map" topic from EasyMotion Studio on line help.

Related functions: TS_SetBuffer

Associated examples:

3.3.12 Miscellaneous

3.3.12.1 TS_Execute

Prototype:

BOOL TML_EXPORT TS_Execute(LPCSTR pszCommands);

Arguments:

	Name	Description
Input	pszCommands	String containing the TML source code to be executed.
Output	return	TRUE if no error, FALSE if error

Description: The function executes the TML commands entered in TML source code format (as is send from the Command Interpreter, in EasyMotion Studio/EasySetUp), from a string containing that code. Use this function if you want to send a specific motion sequence, directly written in TML language.

Build a string **pszCommands** containing the source TML code and then call the **TS_Execute** function in order to compile the code and to send on-line the associated TML object commands.

If a compile error occurs, the function returns a FALSE, otherwise it returns TRUE.

Related functions: TS_ExecuteScript

Associated examples: Ex02_DriveStatus, Ex06_ExternalReference, Ex08_PVT

3.3.12.2 TS_ExecuteScript

Prototype:

BOOL TML_EXPORT TS_ExecuteScript(LPCSTR pszFileName);

Arguments:

	Name	Description
Input	pszFileName	The name of the file containing the TML source code to be executed.
Output	return	TRUE if no error, FALSE if error

Description: The function executes TML commands entered in TML source code format (as is sent from the Command Interpreter in EasyMotion Studio/EasySetUp) from a script file. Use this function if you want to send a specific motion sequence, directly written in TML language.

Define a data file **pszFileName** containing the source TML code you want to send to the drive and then call the **TS_ExecuteScript** function in order to compile the code and to send on-line the associated TML object commands.

If a compile error occurs, the function returns a FALSE, otherwise it returns TRUE.

Related functions: TS_Execute

Associated examples:

3.3.12.3 TS_GetOutputOfExecute

Prototype:

BOOL TML_EXPORT TS_GetOutputOfExecute(LPSTR pszOutput, int nMaxChars);

Arguments:

	Name	Description
Input	pszOutput	String containing the TML source code generated at the last library function call.
Output	return	TRUE if no error, FALSE if error

Description: The function returns the TML output source code of the last previously executed TML_LIB library function call. Use this function if you want to examine the binary code of the TML commands that are generated when you call one of the functions of the TML_LIB library.

The binary code is returned in the **pszOutput** string. Set the maximum number of characters to be returned as the value of the **nMaxChars** parameter.

Related functions: TS_Execute

Associated examples:

3.3.13 Data logger

3.3.13.1 TS_SetupLogger

Prototype:

BOOL TML_EXPORT TS_SetupLogger(WORD wLogBufferAddr, WORD wLogBufferLen, WORD* arrayAddresses, WORD countAddr, WORD period);

Arguments:

	Name	Description
Input	wLogBufferAddr	The address of the logger buffer in drive/motor memory, where data will be stored during logging
	wLogBufferLen	The length in words of the logger buffer
	arrayAddresses	Pointer to the array containing the drive/motor memory addresses to be logged
	countAddr	The number of memory addresses to be logged
	period	Time interval between two consecutive data logging expressed in TML time units
Output	return	TRUE if no error, FALSE if error

Description: The function sets the parameters of the data logger on the active axis. Use this function if you want to perform data logging on the drive/motor during the motion execution and analyze it in the PC application.

Set the **wLogBufferAddress** parameter with the starting address of the drive RAM memory data buffer where a number of **wLogBufferLenlength** data points of logged data will be stored.

The addresses of TML data logged are stored in an array of length **countAddr**. Parameter **arrayAddresses** points to the beginning of the array with.

*Remark The number of data sets which can be stored will be determined as the integer part of the ratio [**length / countAddr**].*

The parameter **period** sets how often the TML data is logged. The period can have any value between 1 and 7FFF.

Remark: Be careful when using the data logger functions! Incorrect settings related to data logger buffer location and size may lead to improper operation of the drive, with unpredictable results.

Related functions: TS_StartLogger, TS_UploadLoggerResults, TS_CheckLoggerStatus

Associated examples: Ex09_Logger

3.3.13.2 TS_StartLogger

Prototype:

BOOL TML_EXPORT TS_StartLogger(WORD wLogBufferAddr, BYTE type);

Arguments:

	Name	Description
Input	wLogBufferAddr	The address of the logger buffer in drive/motor memory, where data will be stored during logging
	type	Specifies when the logging occurs
Output	return	TRUE if no error, FALSE if error

Description: The function starts the data logger on the active axis. The function may be called only after the initialization of the data logger with the **TS_SetupLogger** function.

Use the parameter **type** to set if the data logging process must be done in the slow control loop (**type = LOGGER_SLOW**), or in the fast control loop (**type = LOGGER_FAST**).

Related functions: TS_SetupLogger, TS_UploadLoggerResults, TS_CheckLoggerStatus

Associated examples: Ex09_Logger

3.3.13.3 TS_CheckLoggerStatus

Prototype:

BOOL TML_EXPORT TS_CheckLoggerStatus(WORD& status);

Arguments:

	Name	Description
Input	wLogBufferAddr	The address of the logger buffer in drive/motor memory, where data will be stored during logging
Output	status	Number of points still remaining to capture; if it is 0, the logging is completed
	return	TRUE if no error, FALSE if error

Description: The function checks the data logger status on the active axis. Use this function in order to check if the data logging process is still running, or if the data logging process was ended. The function returns the **status** parameter, whose value indicates how many points are still to be captured. If **status = 0** the data logging process is finished.

The function may be called only after the start of the logging process with the **TS_StartLogger** function.

Related functions: TS_ SetupLogger, TS_StartLogger, TS_UploadLoggerResults

Associated examples: Ex09_Logger

3.3.13.4 TS_UploadLoggerResults

Prototype:

BOOL   TML_EXPORT   TS_UploadLoggerResults(WORD   wLogBufferAddr,   WORD* arrayValues, WORD& countValues);

Arguments:

	Name	Description
Input	wLogBufferAddr	The address of the logger buffer in drive/motor memory, where data will be stored during logging
	arrayValues	Pointer to the array where the uploaded data is stored on the PC
	countValues	The size of arrayValues, expressed in WORDs
Output	countValues	The number of uploaded data
	return	TRUE if no error, FALSE if error

Description: The function uploads the data logged from the active axis. Use this function to upload the data stored during the data logger execution. Before calling the function, you must declare a data buffer in the PC program, starting at the **arrayValues** address, with a size equal to the **countValues** parameter.

The **TS_UploadLoggerResults** function will fill the **arrayValues** data buffer with the data transferred from the drive, and will also return the actual number of transferred data words, in the **countValues** parameter. Once the data is transferred, you can use it for data analysis, graphical representation.

Remark:
1. *Prior uploading the data logged, call function TS_CheckLoggerStatus to test the end of data logging.*
2. *The number of data sets which were stored will be determined as the integer part of the ratio [**length / countAddr**] where **length** and **countAddr** are setup parameters defined when calling the **TS_SetupLogger** function*

The uploaded data is stored in consecutive data sets, i.e. the first set of **countAddr** words will contain the first logged point for the selected variables, the second set of **countAddr** words will contain the second logged point for the selected variables, and so on. The following table illustrates this data structure for an example of **4 logged variables**.

Data WORD	Meaning
1	**Variable 1, point 1**
2	Variable 2, point 1
3	Variable 3, point 1
4	Variable 4, point 1
5	**Variable 1, point 2**
6	Variable 2, point 2
7	Variable 3, point 2
…	…

Related functions: TS_ SetupLogger, TS_StartLogger, TS_ CheckLoggerStatus

Associated examples: Ex09_Logger

This page is empty

4 Examples

This chapter presents a collection of applications which use the functions of the **TML_LIB** library to provide you a first, basic insight about using the **TML_LIB** library to implement your motion control applications.

Note that most of these examples contain function calls to TML_LIB functions, and are based on the hypothesis that the setup data is already downloaded into the non-volatile memory of the drive so that you'll directly start sending motion commands from the PC to the drive.

The examples are built for configurations with Technosoft drive **IBL2403-CAN with brushless motor**.

Remarks:

1. *Prior running the examples, generate the setup data and modify the examples to accommodate the IO configuration of your drive/motor. Also, the examples are switching between position control and speed control, therefore during the Drive Setup phase enable all 3 control loops, current, speed and position, and tune the controllers.*

2. *The examples for Microsoft Windows platform, require the Working Directory to be set to the **examples** folder of the TML_lib, by default C:\Program Files\Technosoft\TML_LIB\examples\. For details about setting the Working Directory read the development environment online help.*

3. *For projects developed under Delphi and C#, the TML_lib.dll and tmlcomm.dll must be preset in the Output Directory of the project.*

4. *Most TML_LIB functions return a Boolean TRUE if the function executed correctly, and a FALSE if any error occurred (incorrect parameters, failed operation at the PC level). Normally, you must check after each function call if there was an error or not. In case of error use function TS_GetLastTextError to obtain a description of the error occurred.*

4.1 Start Up

The example details the steps required to build the host application based on TML_lib. Before starting to build the host application you must setup the drive/motor accordingly with your application. The drive/motor setup is done using EasySetUp/EasyMotion Studio. When the setup is finished the host application must include the basic functionality:

1. Open communication channel using the **TS_OpenChannel**. The TML_lib library supports RS232, RS485, CAN and Ethernet communication.

2. Load the setup data for each axis controlled from the host with **TS_LoadSetup** function. The setup data must be generated from EasySetUp/EasyMotion Studio. The setup information is included in a *.t.zip file and is used by the library to validate your commands; it doesn't contain the setup data downloaded in the drive non volatile memory.

3. Associate each axis with the setup information using the **TS_SetupAxis**. An axis is defined by its Axis ID and the setup information. The Axis ID is assigned to a drive/motor in the Drive setup dialog.

4. Select the destination of the commands sent from the host. For single axis applications the selection of the destination must be done once, after the axis setup. In multiaxis applications the **TS_SelectAxis** must be called every time the messages' destination is changed.

5. Call **TS_DriveInitialization** function to check the integrity of the setup data downloaded in the non volatile memory of the drive. The setup data validation is performed automatically by the drive when it is powered. The TS_DriveInitialization function also signals the end of the drive initialization.

6. Enable the power stage of the drive with **TS_Power** function.

7. Send the motion commands required by your application using the functions included in the TML_lib.

8. When the application is finished, disable the power stage of the drive by calling the **TS_Power** function.

9. Close the communication channels using the **TS_CloseChannel** function. To close the current communication channel pass -1 to the TS_CloseChannel.

4.2 Drive status

The drive's/motor's key information is grouped in 2 status registers: Status Register (32-bit) and Motion Error Register (16-bit). The host can monitor the status of the drive/motor by:

- Requesting periodically the values of the status registers (SRL, SRH and MER) using the TS_ReadStatus function.

- Enable the drive to send automatically its status. The message transmission is triggered by conditions which change the status registers or the error register. The host selects the bits from the registers that will trigger a message, via 3 masks one for each register: SRL_MASK, SRH_MASK and MER_MASK.

  The host analyse the message content in the user callback function, called automatically by the TML_lib when the host receives a message from the host.

4.3 Error handling

When an error occurs, the drive enters in the fault status. In the fault status the power stage is disabled, the the MER register signals the erros occurred and bit 15 from the SRH is set high to signal the fault state.

The normal operation of the drive can be restored by:

- calling the TS_ResetFault function. The function call must be followed by the TS_Power in order to enable the power stage of the drive. If the error persists then the drive will return to the faulte state.

- reseting the drive using the TS_Reset function when the application depends on special routines to be executed, i.e homing routines. After a reset command the communication with the drive is disabled until the reset routines end. Hence the host application should add a delay before restoring the communication with the drive/motor. After the reset the drive communicates using the default baudrate, i.e. 9600bps for serial communication and 500kbps for CAN communication. If the host application was using a different baudrate before the reset then use MSK_SetBaudRate function to first set baudrate to the default values and then set it to the initial value.

4.4 Basic move

Technosoft drives/motors can execute a broad range of motion profiles. The example covers only the basic motion profiles like

- positionings with trapezoidal speed profile
- positionings with Scurve speed profiles
- velocity profiles

*Remark: The example requires all control loops to be enabled. The control scheme is selected in the **Drive Setup** dialog. To enable all control loops first select Position in the Control mode, then open the **Advanced** dialog. In the Advanced dialog select the **Close position, speed and current loop** option and press the OK button.*

4.5 Homing

Some application required a well defined starting point. This condition is achieved by calling a homing routine before beginning the main motion. The homing routine can be built:

- in the host application, the host calling the TML_lib functions required, or
- in EasyMotion Studio, using the TML language, then downloaded in the drive's memory and launched from the host application.

The first part of the example calls a homing routine developed in EasyMotion Studio and downloaded in the drive's non volatile memory. Before running the example:

- start EasyMotion Studio and establish the communication with the drive/motor
- restore the **TML_lib_examples** project from the TML_lib_examples.m.zip archive using the **Project | Restore…** menu command
- select the Ex05_Homing application and use the **Application | Motion | Download Program** menu command to download the homing routine in the non volatile memory of the drive/motor

After downloading the homing routine you can call it from the host application using the TS_CancelableCALL_Label.

Remark: *When a TML function is called with the TS_CancelableCALL_Label function, the SRL.8 bit is set. The bit is reset when the function is finished. You should monitor the value of this bit to check the status of the homing routine.*

In the second part of the example, a similar routine is executed by calling the functions included in the TML_lib library.

4.6 External reference

Thechnosoft drives/motors are capable to use external reference signals provided by other devices. There are 3 types of external references:

- Analogue – read by the drive/motor via a dedicated analogue input (10-bit resolution)
- Digital – computed by the drive/motor from:
    - Pulse & direction signals
    - Quadrature signals like A, B signals of an incremental encoder
- Online – received online via a communication channel from a host and saved in a dedicated TML variable

The example is split in 4 parts, one for each type of external reference and in the fourt the analog reference is used to compute a speed command.

4.7 Multiaxes

In multiaxes mode one of the drives acts as master providing the reference for the other drives. The drives/motors can operate in electronic gearing or electronic camming.

When set as **master**, in electronic gearing, the drive/motor sends its position via a multi-axis communication channel, like the CANbus. When set as **slave**, the drive/motor follows the master position with a programmable gear ratio.

In electronic camming the drive/motor set as **master**, sends its position via a multi-axis communication channel, and the drive/motor set as **slave** executes a cam profile function of the master 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 first part of the example presents the steps required to set 2 drives in electronic gearing mode, one as master and the other as slave. The second part of the example illustrates the use of the cam table files and the phases required to set the electronic camming mode.

4.8 PVT – multithreading

In the PVT motion mode the built-in reference generator computes a positioning path using a series of points. Each point specifies the desired **P**osition, **V**elocity and **T**ime, i.e. contains a **PVT** data. Between the PVT points the reference generator performs a 3rd order interpolation.

The example walks through the steps required to set the PVT mode and use separates threads for each axis. The drives follow a complex path composed from lines and circles. The main thread computes the PVT points, then creates a thread for each axis. In the treads the host sends the PVT points and handles the PVT status received from the drives. When the trajectory is completed each thread closes the communication channel.

4.9 Logger

The **Data Logger** is an advanced graphical analysis tool, allowing you to do data acquisitions on any variable of your drive / motor and plot the results.

Please note that the uploaded data is stored alternatively. Also you have to take in consideration the type of the data received (integer, long, fixed) especially for fixed (16bit integer part. 16 bit fixed part) variables which must be converted from a 32-bit integer to float.

4.10 Event handling

An event is a programmable condition, which once set, is monitored for occurrence. Only a single event can be programmed at a time. You can do the following actions in relation with an event:

- Change the motion mode and/or the motion parameters, when the event occurs
- Stop the motion when the event occurs
- Wait for the programmed event to occur

The host application can:

- tests continuously the event status, and waits until the event occurs. There is a drawback of this situation, if the event will not occur, due to some unexpected conditions the program hangs-up in an internal loop waiting for the event to occur.
- Check periodically if the event occurred using the TS_CheckEvent function. In this way, you can detect if the event does not occur and eventually exit from the test loop after a given time period.

4.11 I/O handling

Each Technosoft drive/motor has a specific number of digital inputs and outputs. Some drives/motors include I/O lines that may be used either as inputs or as outputs. In these cases before using these lines, you need to specify how you want to use them by calling the TS_SetupInput and TS_SetupOutput functions.

Remarks:
- *Read carefully the drive/motor user manual to find which I/O lines are available.*
- *You need to set an I/O line as input or output, only once, after power on*

The I/O lines can be controlled individually or simultaneously; the example covers both cases.

4.12 Distributed tasks

The emebeded intelligence of the drive/motor allows you to distribute the application between the host and the drives/motors in complex multi-axis applications. Thus, the host, instead of sending commands for each step of an axis movement, it can simply trigger the execution of a TML function, stored on the drive, and monitor the drive status. This approach allows the host to focus on system functionality, leaving the drive to handle the motion tasks.

The steps for building a distributed application are:

1. implement the TML functions using EasyMotion Studio
2. download the TML code on the drive:

    a. from EasyMotion Studio in the non volatile memory
    b. from the host application using the executable file (COFF) resulted after compiling the TML code. The COFF file can be downloaded with TS_DownloadProgram function.
    c. from the host application using the software file (sw) generated from EasyMotion Studio.

3. export the setup data for TML_lib . The setup data generated for TML_lib will contain the name of the TML functions created in EasyMotion Studio.
4. trigger, from the host, the execution of the TML functions by calling the TS_CancelableCALL_Lable function.
5. monitor the function execution status by requesting periodically the status registers or by enabling the drive to send its status automatically.

The example shows 2 possible ways of executing TML code developed using the EasyMotion Studio.

This page is empty

Appendix A Axis identification

The data exchanged on the communication channel 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 group of axes. Each drive/motor has its own 8-bit Axis ID and Group ID. This information is stored in TML variable **AAR**.

Remarks:

1. *The Axis ID of a drive/motor must be **unique** and is set during the drive/motor setup phase with **EasySetUp/EasyMotion Studio**. The possible values for Axis ID are between 1 and 255.*
2. *In the TechnoCAN communication protocol the Axis ID is interpreted as modulo 32.*
3. *Use TS_GetIntVariable to read the value of the Axis ID and Group ID from the AAR (uint@0x030C) TML variable.*

The Group ID represents a way to identify a group of axes, for a multicast transmission. This feature allows sending 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.

Remarks:

1. *A drive/motor belongs, by default, to the group ID = 1.*
2. *The TechnoCAN protocols supports up to 5 groups, possible Group ID values: 1 to 5*

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

Table 4.1Definition 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)

This page is empty

Appendix B 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 configuration, use:

- **Help | Help Topics | Setup Data Management | Internal Units and Scaling Factors** menu command in EasySetUp
- **Help | Help Topics | Application Programming Internal Units and Scaling Factors** menu command in EasyMotion Studio

*Important: The **Internal Units and Scaling Factors** 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!*

This page is empty

Appendix C CAM files format

The cam tables are arrays of X, Y points, where X is the cam input i.e. the master position and Y is the cam output i.e. the slave position. The X points are expressed in the master internal position units, while the Y points are expressed in the slave internal position units. Both X and Y points 32-bit long integer values. The X points must be positive (including 0) and equally spaced at: 1, 2, 4, 8, 16, 32, 64 or 128 i.e. having the interpolation step a power of 2 between 0 and 7. The maximum number of points for one cam table is 8192.

As cam table X points are equally spaced, they are completely defined by two data: the Master start value or the first X point and the Interpolation step providing the distance between the X points. This offers the possibility to minimize the cam size, which is saved in the drive/motor in the following format:

- 1st word (1 word = 16-bit data)
- Bits 15-13 - the power of 2 of the interpolation step. For example, if these bits have the binary value 010 (2), the interpolation step is 22 = 4, hence the master X values are spaced from 4 to 4: 0, 4, 8, 12, etc.
- Bits 12-0 - the length -1 of the table. The length represents the number of points
- 2nd and 3rd words: the Master start value (long), expressed in master position units. 2nd word contains the low part, 3rd word the high part
- 4th and 5th words: Reserved. Must be set to 0
- Next pairs of 2 words: the slave Y positions (long), expressed in position units. The 1st word from the pair contains the low part and the 2nd word from the pair the high part
- Last word: the cam table checksum, representing the sum modulo 65536 of all the cam table data except the checksum word itself

This page is empty

Appendix D Package contents of TML_LIB for Microsoft Windows

Details the package contents of TML_LIB for Microsoft Windows.

Directory	Files	Description
Root directory	P091.040.UM.PDF	The PDF file of the TML_LIB user manual (this document)
	ChangeLog.txt	Contains the release information
lib	TML_lib.dll	TML_LIB DLL library file
	tmlcomm.dll	TML_lib communication module file
	TML_lib.lib	Import library of TML_LIB
	tmlcomm.lib	Import library of tmlcomm
	TML_lib-borlandc.lib	import library variant of TML_LIB for Borland C++ linker projects
	tmlcomm-borlandc.lib	import library of tmlcomm for Borland C++ projects
lib-multithread	TML_lib.DLL	DLL variant with multithread capabilities
	tmlcomm.DLL	communication module variant with multithread capabilities
	TML_lib.lib	lib file for TML_lib variant with multithread capabilities
	tmlcomm.lib	lib file for tmlcomm.dll with multithread
	TML_lib-borlandc.lib	import library of TML_lib variant with multithread capabilities for Borland C++
	tmlcomm-borlandc.lib	import library of tmlcomm.dll with multithread for Borland C++
include	TML_lib.h	Header file for VC++ applications
	TML_lib.bas	Header file for VB applications
	TML_lib.pas	Header file for Pascal applications
	TML_lib.cs	TML_lib classes for C#
examples	Ex05_Homing.t.zip	Setup data used in example Ex05_Homing
	Ex12_cmd_variable.t.zip	Setup data used in example Ex12_DistributedTasks
	Ex12_cmd_variable.out	Executable file (COFF) used in example Ex12_DistributedTasks
	Ex12_COFF2RAM.t.zip	Setup data used in example Ex12_DistributedTasks
	Ex12_COFF2RAM.out	Executable file (COFF) used in example Ex12_DistributedTasks
	Exx_setup_file_ID1.t.zip	Setup data used in the rest of the examples
	Exx_setup_file_ID2.t.zip	Setup data used in example Ex07_MultiAxes
	ExampleCam.cam	Cam table generated from EasyMotion Studio. The cam table is used in example Ex07_Multiaxes
	ExampleCam.cam	Cam table in text format. Can be imported in EasyMotion Studio to generate the *.cam file
	TML_lib_examples	Archive of TML project used for examples Ex05_Homing and Ex12_DistributedTasks.

examples\Cdemo[1]	C projects with all the examples	Complete projects for Visual C++ and Borland C++ Builder implementing the examples from Chapter 4.
examples\C#demo[1]	C# projects with all the examples	Complete projects for C# implementing the examples from Chapter 4.
examples\VBdemo[2]	VB project and all examples in Visual Basic	A complete Visual Basic project implementing equivalent examples of the examples presented Chapter 4. **The example requires the single thread variant of the TML_lib.**
examples\DELPHIdemo[2]	Delphi project and all examples in Pascal	A complete Delphi project implementing equivalent examples of those presented in Chapter 4.
examples\VCvirtRS232	Visual C project of the virtual serial driver	A complete Visual C project of a communication driver example for the virtual serial communication

---

[1] The examples are available for 32-bit and 64-bit versions of the TML_lib for Microsoft Windows platforms
[2] The examples are available only for the 32-bit version of the TML_lib for Microsoft Windows platforms

Appendix E Package contents of TML_LIB for Linux

Details the package contents for TML_LIB for Linux.

Directory	Files	Description
/usr/include	TML_lib.h	Header file for TML_lib library
	tmlcomm.h	Header file for tmlcomm library
/usr/lib	libTML_lib.so	TML_LIB library file
	libtmlcomm.so	TML communication library file
/usr/share/doc/TML_lib/	Changelog	Contains the release information
	Check-for-updates_linux_x86	Web page which facilitates the update of the library via Internet
	License	License agreement for TML_LIB
	P091.040.UM.xxxx.PDF	The PDF file of the TML_LIB user manual (this document)
/usr/share/doc/TML_lib/examples/src/	C examples	Complete C projects implementing the examples from Chapter 4.
/usr/share/doc/TML_lib/examples/TML_LIB_User/	Ex05_Homing.t.zip	Setup data used in example Ex05_Homing
	Ex12_cmd_variable.t.zip	Setup data used in example Ex12_DistributedTasks
	Ex12_cmd_variable.out	Executable file (COFF) used in example Ex12_DistributedTasks
	Ex12_COFF2RAM.t.zip	Setup data used in example Ex12_DistributedTasks
	Ex12_COFF2RAM.out	Executable file (COFF) used in example Ex12_DistributedTasks
	Exx_setup_file_ID1.t.zip	Setup data used in the rest of the examples
	Exx_setup_file_ID2.t.zip	Setup data used in example Ex07_MultiAxes
	ExampleCam.cam	Cam table generated from EasyMotion Studio. The cam table is used in example Ex07_Multiaxes
	ExampleCam.cam	Cam table in text format. Can be imported in EasyMotion Studio to generate the *.cam file
	TML_lib_examples	Archive of TML project used for examples Ex05_Homing and Ex12_DistributedTasks.
/usr/share/doc/TML_lib/peak-linux-drivers-patches	Patches for Peak System CAN-bus devices drivers versions 3.17 and 4.3	

Appendix F TML_LIB.h file

The TML_LIB.h file is the header file containing the prototypes of all TML_LIB functions, as well as all the constants needed to call functions of the library.

This file must be included in any C file that refers functions from TML_LIB. Use the TML_LIB.lib and the TML_LIB.dll to build the corresponding executable file.

```c
#ifndef __TML_LIB_H__
#define __TML_LIB_H__

#if defined(WINDOWS) || defined(WIN32)
#  ifdef _TMLDLL
#    define TML_EXPORT __declspec(dllexport) __stdcall
#  else
#    define TML_EXPORT __declspec(dllimport) __stdcall
#  endif
#else
#  define TML_EXPORT
#endif


#undef BYTE
#undef WORD
#undef DWORD
#undef BOOL
#if defined(WINDOWS) || defined(WIN32)
        typedef unsigned char BYTE;
        typedef unsigned short WORD;
        typedef unsigned long DWORD;
#else
        typedef u_int8_t BYTE;
        typedef u_int16_t WORD;
        typedef u_int32_t DWORD;
#endif
typedef int BOOL;
#ifndef FALSE
        #define FALSE  0
#endif
#ifndef TRUE
        #define TRUE   1
#endif

typedef const char* LPCSTR;
typedef char* LPSTR;

//supported CAN protocols
#define PROTOCOL_TMLCAN        0x00 /*use TMLCAN protocol (default, 29-
bit identifiers)*/
```

```
#define PROTOCOL_TECHNOCAN                              0x80 /* use
TechnoCAN protocol (11-bit identifiers)*/
#define PR0TOCOL_MASK                                   0x80 /*this bits
are used for specifying CAN protocol through nChannelType param of
MSK_OpenComm function*/


/***** supported CAN devices ****************************
CHANNEL_IXXAT_CAN - see http://www.ixxat.com
CHANNEL_SYS_TEC_USBCAN - see www.systec-electronic.com
CHANNEL_ESD_CAN - see http://www.esd-electronics.com
CHANNEL_PEAK_SYS_PCAN_* - see http://www.peak-system.com
CHANNEL_LAWICEL_USBCAN - see http://www.canusb.com
**********************************************************/


/*Constants used as values for 'OpenChannel' parameters*/
#define CHANNEL_RS232               0
#define CHANNEL_RS485               1
#define CHANNEL_IXXAT_CAN           2
#define CHANNEL_SYS_TEC_USBCAN      3
#define CHANNEL_PEAK_SYS_PCAN_PCI   4
#define CHANNEL_ESD_CAN             5
#define CHANNEL_PEAK_SYS_PCAN_ISA   6
#define CHANNEL_PEAK_SYS_PCAN_PC104    CHANNEL_PEAK_SYS_PCAN_ISA /* Same
with PCAN_ISA*/
#define CHANNEL_PEAK_SYS_PCAN_USB   7
#define CHANNEL_PEAK_SYS_PCAN_DONGLE 8
#define CHANNEL_LAWICEL_USBCAN      9
#define CHANNEL_VIRTUAL_SERIAL      15
#define CHANNEL_XPORT_IP            16


/*Constant used for host ID*/
#define HOST_ID         1


/*Constants used as values for 'Logger' parameters*/
#define LOGGER_SLOW             1
#define LOGGER_FAST             2


/*Constants used as values for 'MoveMoment' parameters*/
#define UPDATE_NONE                     -1
#define UPDATE_ON_EVENT                 0
#define UPDATE_IMMEDIATE        1


/*Constants used for 'ReferenceType' parameters*/
#define REFERENCE_POSITION              0
#define REFERENCE_SPEED                         1
#define REFERENCE_TORQUE                2
#define REFERENCE_VOLTAGE               3



/*Constants used for   EnableSuperposition*/
#define SUPERPOS_DISABLE        -1
```

```
#define SUPERPOS_NONE            0
#define SUPERPOS_ENABLE                1


/*Constants used for   PositionType*/
#define ABSOLUTE_POSITION        0
#define RELATIVE_POSITION        1


/*Constants used for   EnableSlave*/
#define SLAVE_NONE                                       0
#define SLAVE_COMMUNICATION_CHANNEL            1
#define SLAVE_2ND_ENCODER                               2


/*Constants used for ReferenceBase*/
#define FROM_MEASURE    0
#define FROM_REFERENCE 1


/*Constants used for DecelerationType*/
#define S_CURVE_SPEED_PROFILE            0
#define TRAPEZOIDAL_SPEED_PROFILE        1


/*Constants used for IOState*/
#define IO_HIGH         1
#define IO_LOW          0


/*Constants used for TransitionType*/
#define TRANSITION_HIGH_TO_LOW          -1
#define TRANSITION_DISABLE                      0
#define TRANSITION_LOW_TO_HIGH          1


/*Constants used for IndexType*/
#define INDEX_1          1
#define INDEX_2          2


/*Constants used for LSWType*/
#define LSW_NEGATIVE            -1
#define LSW_POSITIVE            1


/*Constants used for TS_Power; to activate/deactivate teh PWM commands*/
#define POWER_ON               1
#define POWER_OFF              0


/*Constants used as inputs parameters of the I/O functions*/
#define INPUT_0   0
#define INPUT_1   1
#define INPUT_2   2
#define INPUT_3   3
#define INPUT_4   4
#define INPUT_5   5
#define INPUT_6   6
#define INPUT_7   7
#define INPUT_8   8
```

```
#define INPUT_9        9
#define INPUT_10       10
#define INPUT_11       11
#define INPUT_12       12
#define INPUT_13       13
#define INPUT_14       14
#define INPUT_15       15
#define INPUT_16       16
#define INPUT_17       17
#define INPUT_18       18
#define INPUT_19       19
#define INPUT_20       20
#define INPUT_21       21
#define INPUT_22       22
#define INPUT_23       23
#define INPUT_24       24
#define INPUT_25       25
#define INPUT_26       26
#define INPUT_27       27
#define INPUT_28       28
#define INPUT_29       29
#define INPUT_30       30
#define INPUT_31       31
#define INPUT_32       32
#define INPUT_33       33
#define INPUT_34       34
#define INPUT_35       35
#define INPUT_36       36
#define INPUT_37       37
#define INPUT_38       38
#define INPUT_39       39

#define OUTPUT_0       0
#define OUTPUT_1       1
#define OUTPUT_2       2
#define OUTPUT_3       3
#define OUTPUT_4       4
#define OUTPUT_5       5
#define OUTPUT_6       6
#define OUTPUT_7       7
#define OUTPUT_8       8
#define OUTPUT_9       9
#define OUTPUT_10      10
#define OUTPUT_11      11
#define OUTPUT_12      12
#define OUTPUT_13      13
#define OUTPUT_14      14
#define OUTPUT_15      15
#define OUTPUT_16      16
#define OUTPUT_17      17
```

```
#define OUTPUT_18        18
#define OUTPUT_19        19
#define OUTPUT_20        20
#define OUTPUT_21        21
#define OUTPUT_22        22
#define OUTPUT_23        23
#define OUTPUT_24        24
#define OUTPUT_25        25
#define OUTPUT_26        26
#define OUTPUT_27        27
#define OUTPUT_28        28
#define OUTPUT_29        29
#define OUTPUT_30        30
#define OUTPUT_31        31
#define OUTPUT_32        32
#define OUTPUT_33        33
#define OUTPUT_34        34
#define OUTPUT_35        35
#define OUTPUT_36        36
#define OUTPUT_37        37
#define OUTPUT_38        38
#define OUTPUT_39        39

/*Constants used for the register for function TS_ReadStatus*/
#define REG_MCR          0
#define REG_MSR          1
#define REG_ISR          2
#define REG_SRL          3
#define REG_SRH          4
#define REG_MER          5

/*Constants used to select or set the group*/
#define GROUP_0          0
#define GROUP_1          1
#define GROUP_2          2
#define GROUP_3          3
#define GROUP_4          4
#define GROUP_5          5
#define GROUP_6          6
#define GROUP_7          7
#define GROUP_8          8

/*Special parameter values*/
#define FULL_RANGE       0
#define NO_VARIATION     0

/************************************************************************
Callback function used by client application for handling unsolicited
messages which this driver receives in unexpected places
************************************************************************/
```

```
#if defined(WINDOWS) || defined(WIN32)
    typedef void (__stdcall *pfnCallbackRecvDriveMsg)(WORD wAxisID, WORD
wAddress, long Value);
#else
    typedef void (*pfnCallbackRecvDriveMsg)(WORD wAxisID, WORD wAddress,
long Value);
#endif

#ifdef __cplusplus
extern "C" {
#endif

LPCSTR TML_EXPORT TS_GetLastErrorText(void);
/************************************************************************
 Function: Returns a text related to the last occurred error when one of
the library functions
           was called.
 Input arguments:
        -
 Output arguments:
        return: A text related to the last occurred error
***********************************************************************/


/******************************************************************/
/*****************Parametrization********************************/
/******************************************************************/


int TML_EXPORT TS_LoadSetup(LPCSTR setupPath);
/************************************************************************
 Function: Load setup information from a zip archive or a directory
containing setup.cfg and variables.cfg files.
 Input arguments:
        setupPath:              path to the zip archive or directory that
contains setup.cfg and variables.cfg of the given setup
 Output arguments:
        return:         >=0 index of the loaded setup; -1 if error
***********************************************************************/


/******************************************************************/
/***************** Communication channels *********************/
/******************************************************************/


int TML_EXPORT TS_OpenChannel(LPCSTR pszDevName, BYTE btType, BYTE
nHostID, DWORD baudrate);
/************************************************************************
 Function: Open a communication channel.
 Input arguments:
        pszDevName:     communication device name
                        RS232, RS485 and CHANNEL_LAWICEL_USBCAN: COM
port number or COM port name ("1","2","3"... -> "COM1", "COM2",
"COM3"...)
                        CHANNEL_IXXAT_CAN: "1" .. "4"
```

CHANNEL_PEAK_SYS_PCAN_PCI: "1" or "2"
CHANNEL_LAWICEL_USBCAN: "" for the first device found or the serial number of the device
CHANNEL_XPORT_IP: "IP" or "hostname"
btType: channel type (CHANNEL_*) with an optional protocol (PROTOCOL_*, default is PROTOCOL_TMLCAN)
nHostID: Is the address of your PC computer. A value between 1 and 255
For RS232: axis ID of the drive connected to the PC serial port (usually 255)
For RS485 or CAN devices: must be an unused axis ID! It is the address of your PC computer on
the RS485 network.
For XPORT: "IP:port"
BaudRate: Baud rate
serial ports: 9600, 19200, 38400, 56000 or 115200
CAN devices: 125000, 250000, 500000, 1000000
 Output arguments:
return: channel's file descriptor or -1 if error
***/

```
BOOL TML_EXPORT TS_SelectChannel(int fd);
```
/***
 Function: Select active communication channel. If you use only one channel there is no need to call this function.
 Input arguments:
fd: channel file descriptor (-1 means selected communication channel)
 Output arguments:
return: TRUE if no error; FALSE if error
**/

```
#ifdef __cplusplus
void TML_EXPORT TS_CloseChannel(int fd = -1);
#else
void TML_EXPORT TS_CloseChannel(int fd);
#endif
```
/***
 Function: Close the communication channel.
 Input arguments:
  fd: channel file descriptor (-1 means selected communication channel)
**/

/**/
/*******************Drive Administration ************************/
/**/

```
BOOL TML_EXPORT TS_SetupAxis(BYTE axisID, int idxSetup);
```
/***

```
    Function: Select setup configuration for the drive with axis ID.
    Input arguments:
          axisID:          axis ID. It must be a value between 1 and 255;
          idxSetup:        Index of previously loaded setup,
    Output arguments:
          return:          TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SelectAxis(BYTE axisID);
/***********************************************************************
    Function: Selects the active axis.
    Input arguments:
          axisID:          The ID of the axis to become the active one. It
must be a value between 1 and 255;
                                        For RS485/CAN communication,
this value must be different than nHostID parameter
                                        defined at TS_OpenChannel
function call.
    Output arguments:
          return:          TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SetupGroup(BYTE groupID, int idxSetup);
/***********************************************************************
    Function: Select setup configuration for the drives within group.
    Input arguments:
          groupID:                 group ID. It must be a value between 1
and 8;
          idxSetup:                Index of previously loaded setup,
    Output arguments:
          return:          TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SelectGroup(BYTE groupID);
/***********************************************************************
    Function: Selects the active group.
    Input arguments:
          groupID:                 The ID of the group of axes to become the
active ones. It must be a value
                                        between 1 and 8.
    Output arguments:
          return:          TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SetupBroadcast(int idxSetup);
/***********************************************************************
    Function: Select setup configuration for all drives on the active
channel.
    Input arguments:
          idxSetup:                Index of previously loaded setup,
    Output arguments:
          return:          TRUE if no error; FALSE if error
```

```
/************************************************************************/

BOOL TML_EXPORT TS_SelectBroadcast(void);
/************************************************************************
 Function: Selects all axis on the active channel.
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


BOOL TML_EXPORT TS_Reset(void);
/************************************************************************
 Function: Resets selected drives.
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


BOOL TML_EXPORT TS_ResetFault(void);
/************************************************************************
 Function: This function clears most of the errors bits from Motion
Error Register (MER).
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


BOOL TML_EXPORT TS_Power(BOOL Enable);
/************************************************************************
 Function: Controls the power stage (ON/OFF).
 Input arguments:
        Enable:         TRUE -> Power ON the drive; FALSE -> Power OFF
the drive
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_ReadStatus(short SelIndex, WORD& Status);
#else
BOOL TML_EXPORT TS_ReadStatus(short SelIndex, WORD* Status);
#endif
/************************************************************************
 Function: Returns drive status information.
 Input arguments:
        SelIndex:
                REG_MCR -> read MCR register
                REG_MSR -> read MSR register
                REG_ISR -> read ISR register
                REG_SRL -> read SRL register
                REG_SRH -> read SRH register
                REG_MER -> read MER register
 Output arguments:
        Status:drive status information (value of the selected register)
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


BOOL TML_EXPORT TS_Save(void);
```

```
/************************************************************************
 Function: Saves actual values of all the parameters from the
drive/motor working memory into
            the EEPROM setup table.
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_UpdateImmediate(void);
/************************************************************************
 Function: Update the motion mode immediately.
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_UpdateOnEvent(void);
/************************************************************************
 Function: Update the motion mode on next event occurence.
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_SetPosition(long PosValue);
/************************************************************************
 Function: Set actual position value.
 Input arguments:
        PosValue:       Value at which the position is set
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_SetCurrent(short CrtValue);
/************************************************************************
 Function: Set actual current value.
 Input arguments:
        CrtValue:       Value at which the motor current is set
                                      REMARK: this command can be used
only for step motor drives
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_SetTargetPositionToActual(void);
/************************************************************************
 Function: Set the target position value equal to the actual position
value.
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_SetIntVariable(LPCSTR pszName, short value);
```

```
/***********************************************************************
 Function: Writes an integer type variable to the drive.
 Input arguments:
        pszName:        Name of the variable
        value: Variable value
 Output arguments:
        return:         TRUE if no error; FALSE if error
***********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_GetIntVariable(LPCSTR pszName, short& value);
#else
BOOL TML_EXPORT TS_GetIntVariable(LPCSTR pszName, short* value);
#endif
/***********************************************************************
 Function: Reads an integer type variable from the drive.
 Input arguments:
        pszName:        Name of the variable
 Output arguments:
        value: Variable value
        return:         TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SetLongVariable(LPCSTR pszName, long value);
/***********************************************************************
 Function: Writes a long integer type variable to the drive.
 Input arguments:
        pszName:        Name of the variable
        value: Variable value
 Output arguments:
        return:         TRUE if no error; FALSE if error
***********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_GetLongVariable(LPCSTR pszName, long& value);
#else
BOOL TML_EXPORT TS_GetLongVariable(LPCSTR pszName, long* value);
#endif
/***********************************************************************
 Function: Reads a long integer type variable from the drive.
 Input arguments:
        pszName:        Name of the variable
 Output arguments:
        value: Variable value
        return:         TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SetFixedVariable(LPCSTR pszName, double value);
/***********************************************************************
 Function: Writes a fixed point type variable to the drive.
 Input arguments:
        pszName:        Name of the variable
```

```
        value: Variable value
 Output arguments:
        return:         TRUE if no error; FALSE if error
********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_GetFixedVariable(LPCSTR pszName, double& value);
#else
BOOL TML_EXPORT TS_GetFixedVariable(LPCSTR pszName, double* value);
#endif
/********************************************************************
 Function: Reads a fixed point type variable from the drive.
 Input arguments:
        pszName:        Name of the variable
 Output arguments:
        value: Variable value
        return:         TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SetBuffer(WORD address, WORD* arrayValues, WORD
nSize);
/********************************************************************
 Function: Download a data buffer to the drive's memory.
 Input arguments:
        address:        Start address where to download the data buffer
        arrayValues:    Buffer containing the data to be downloaded
        nSize: the number of words to download
 Output arguments:
        return:         TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_GetBuffer(WORD address, WORD* arrayValues, WORD
nSize);
/********************************************************************
 Function: Upload a data buffer from the drive (get it from motion
chip's memory).
 Input arguments:
        address:        Start address where from to upload the data
buffer
        arrayValues:    Buffer address where the uploaded data will be
stored
        nSize: the number of words to upload
 Output arguments:
        arrayValues:    the uploaded data
        return:         TRUE if no error; FALSE if error
********************************************************************/


/******************************************************************/
/******************MOTION functions*******************************/
/******************************************************************/
```

```
BOOL TML_EXPORT TS_MoveAbsolute(long AbsPosition, double Speed, double
Acceleration, short MoveMoment, short ReferenceBase);
/************************************************************************
 Function: Move Absolute with trapezoidal speed profile. This function
allows you to program a position profile
           with a trapezoidal shape of the speed.
 Input arguments:
         AbsPosition:   Absolute position reference value
         Speed:         Slew speed; if 0, use previously defined value
         Acceleration:  Acceleration  decceleration; if 0, use previously
defined value
         MoveMoment:
                 UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                 UPDATE_IMMEDIATE -> start moving immediate
                 UPDATE_ON_EVENT -> start moving on event
         ReferenceBase:
                 FROM_MEASURE -> the position reference starts from the
actual measured position value
                 FROM_REFERENCE -> the position reference starts from the
actual reference position value
 Output arguments:
         return:        TRUE if no error; FALSE if error
************************************************************************/

BOOL TML_EXPORT TS_MoveRelative(long RelPosition, double Speed, double
Acceleration, BOOL IsAdditive, short MoveMoment, short ReferenceBase);
/************************************************************************
 Function: Move Relative with trapezoidal speed profile. This function
allows you to program a position profile
           with a trapezoidal shape of the speed.
 Input arguments:
         RelPosition:   Relative position reference value
         Speed:         Slew speed; if 0, use previously defined value
         Acceleration:  Acceleration  decceleration; if 0, use previously
defined value
         MoveMoment:
                 UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                 UPDATE_IMMEDIATE -> start moving immediate
                 UPDATE_ON_EVENT -> start moving on event
         IsAdditive:
                 TRUE -> Add the position increment to the position to
reach set by the previous motion command
                 FALSE -> No position increment is added to the target
position
         ReferenceBase:
                 FROM_MEASURE -> the position reference starts from the
actual measured position value
                 FROM_REFERENCE -> the position reference starts from the
actual reference position value
 Output arguments:
```

```
        return:        TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_MoveVelocity(double Speed, double Acceleration, short
MoveMoment, short ReferenceBase);
/********************************************************************
 Function: Move at a given speed, with acceleration profile.
 Input arguments:
        Speed:         Jogging speed
        Acceleration:  Acceleration  decceleration; if 0, use previously
defined value
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
        ReferenceBase:
                FROM_MEASURE -> the position reference starts from the
actual measured position value
                FROM_REFERENCE -> the position reference starts from the
actual reference position value
 Output arguments:
        return:        TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SetAnalogueMoveExternal (short ReferenceType, BOOL
UpdateFast, double LimitVariation, short MoveMoment);
/********************************************************************
 Function: Set Motion type as using an analogue external reference.
 Input arguments:
        ReferenceType:
                REFERENCE_POSITION -> external position reference
                REFERENCE_SPEED -> external speed reference
                REFERENCE_TORQUE -> external torque reference
                REFERENCE_VOLTAGE -> external voltage reference
        UpdateFast:
                TRUE -> generate the torque reference in the fast
control loop
                FALSE -> generate the torque reference in the slow
control loop
        LimitVariation:
                NO_VARIATION (0) -> the external reference is limited at
the value set in the Drive Setup
                A value which can be an acceleration or speed in
function of the reference type.
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
 Output arguments:
        return:        TRUE if no error; FALSE if error
```

```
*********************************************************************/

#ifdef __cplusplus
BOOL TML_EXPORT TS_SetDigitalMoveExternal (BOOL SetGearRatio = FALSE,
short Denominator = 1, short Numerator = 1, double LimitVariation = 0,
short MoveMoment = 1);
#else
BOOL TML_EXPORT TS_SetDigitalMoveExternal (BOOL SetGearRatio, short
Denominator, short Numerator, double LimitVariation, short MoveMoment);
#endif
/*********************************************************************
 Function: Set Motion type as using a digital external reference. This
function is used only for Positioning.
 Input arguments:
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
        LimitVariation:
                NO_VARIATION (0) -> the external reference is limited at
the value set in the Drive Setup
                A value which can be an acceleration or speed in
function of the reference type.
        SetGearRatio: Set the gear parameters; if TRUE, following
parameters are needed
        Denumerator: Gear master ratio
        Numerator: Gear slave ratio
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************/

#ifdef __cplusplus
BOOL TML_EXPORT TS_SetOnlineMoveExternal (short ReferenceType, double
LimitVariation = 0, double InitialValue = 0., short MoveMoment = 1);
#else
BOOL TML_EXPORT TS_SetOnlineMoveExternal (short ReferenceType, double
LimitVariation, double InitialValue,  short MoveMoment);
#endif
/*********************************************************************
 Function: Set Motion type as using an analogue external reference.
 Input arguments:
        ReferenceType:
                REFERENCE_POSITION -> external position reference
                REFERENCE_SPEED -> external speed reference
                REFERENCE_TORQUE -> external torque reference
                REFERENCE_VOLTAGE -> external voltage reference
        LimitVariation:
                NO_VARIATION (0) -> the external reference is limited at
the value set in the Drive Setup
                A value which can be an acceleration or speed in
function of the reference type.
```

```
                MoveMoment:
                    UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                    UPDATE_IMMEDIATE -> start moving immediate
                    UPDATE_ON_EVENT -> start moving on event
            InitialValue: If non zero, set initial value of EREF
 Output arguments:
            return:         TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_VoltageTestMode(short MaxVoltage, short IncrVoltage,
short Theta0, short Dtheta, short MoveMoment);
/********************************************************************
 Function: Use voltage test mode.
 Input arguments:
            MaxVoltage:     Maximum test voltage value
            IncrVoltage:    Voltage increment on each slow sampling period
            Theta0:         Initial value of electrical angle value
                             Remark: used only for AC motors; set to 0
otherwise
            Dtheta:         Electric angle increment on each slow sampling
period
            MoveMoment:
                    UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                    UPDATE_IMMEDIATE -> start moving immediate
                    UPDATE_ON_EVENT -> start moving on event
 Output arguments:
            return:         TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_TorqueTestMode(short MaxTorque, short IncrTorque,
short Theta0, short Dtheta, short MoveMoment);
/********************************************************************
 Function: Use torque test mode.
 Input arguments:
            MaxTorque:      Maximum test torque value
            IncrTorque:     Torque increment on each slow sampling period
            Theta0:         Initial value of electrical angle value
                             Remark: used only for AC motors; set to 0
otherwise
            Dtheta:         Electric angle increment on each slow sampling
period
            MoveMoment:
                    UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                    UPDATE_IMMEDIATE -> start moving immediate
                    UPDATE_ON_EVENT -> start moving on event
 Output arguments:
            return:         TRUE if no error; FALSE if error
********************************************************************/
```

```
BOOL TML_EXPORT TS_SetGearingMaster(BOOL Group, BYTE SlaveID, short
ReferenceBase, BOOL Enable,
                              BOOL SetSlavePos, short MoveMoment);
/***********************************************************************
 Function: Setup master parameters in gearing mode.
 Input arguments:
        Group
                TRUE -> set slave group ID with value;
                FALSE-> set slave axis ID with SlaveID value;
        SlaveID:        Axis ID in the case that Group is FALSE or a
Group ID when Group is TRUE
        ReferenceBase:
                FROM_MEASURE -> send position feedback
                FROM_REFERENCE -> send position reference
        Enable:         TRUE -> enable gearing operation; FALSE ->
disable gearing operation
        SetSlavePos:    TRUE -> initialize slave(s) with master position
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
 Output arguments:
        return:         TRUE if no error; FALSE if error
***********************************************************************/


BOOL TML_EXPORT TS_SetGearingSlave(short Denominator, short Numerator,
short ReferenceBase,
                                                              short
EnableSlave, double LimitVariation, short MoveMoment);
/***********************************************************************
 Function: Setup slave parameters in gearing mode.
 Input arguments:
        Denominator:    Master gear ratio value
        Numerator:      Slave gear ratio value
        ReferenceBase:
                FROM_MEASURE -> the position reference starts from the
actual measured position value
                FROM_REFERENCE -> the position reference starts from the
actual reference position value
        EnableSlave:
                SLAVE_NONE -> do not enable slave operation
                SLAVE_COMMUNICATION_CHANNEL -> enable operation got via
a communication channel
                SLAVE_2ND_ENCODER -> enable operation read from 2nd
encoder or P&D inputs
        LimitVariation:
                NO_VARIATION (0) -> the external reference is limited at
the value set in the Drive Setup
                A value which can be an acceleration or speed in
function of the reference type.
        MoveMoment:
```

```
                  UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                  UPDATE_IMMEDIATE -> start moving immediate
                  UPDATE_ON_EVENT -> start moving on event
 Output arguments:
        return:          TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_MotionSuperposition(short Enable, short Update);
/**********************************************************************
 Function: enable or disable the superposition of the electronic gearing
mode with a second
                           motion mode
 Input arguments:
        Enable: if 0, disable the Superposition mode
                           if 1, enable the Superposition mode
        Update: if 0, doesn't send UPD command to the drive, in order to
take into account the
                           Superposition mode
                           if 1, sends UPD command to the drive, in order
to take into account the
                           Superposition mode
 Output arguments:
        return:          TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_SetCammingMaster(BOOL Group, BYTE SlaveID, short
ReferenceBase, BOOL Enable, short MoveMoment);
/**********************************************************************
 Function: Setup master parameters in camming mode.
 Input arguments:
        Group
                  TRUE -> set slave group ID with (SlaveID + 256) value;
                  FALSE-> set slave axis ID with SlaveID value;
        SlaveID:         Axis ID in case Group is FALSE, or group mask
otherwise (0 means broadcast)
        ReferenceBase:
                  FROM_MEASURE -> send position feedback
                  FROM_REFERENCE -> send position reference
        Enable:          TRUE -> enable camming operation; FALSE ->
disable camming operation
        MoveMoment:
                  UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                  UPDATE_IMMEDIATE -> start moving immediate
                  UPDATE_ON_EVENT -> start moving on event
 Output arguments:
        return:          TRUE if no error; FALSE if error
**********************************************************************/


#ifdef __cplusplus
```

```
BOOL TML_EXPORT TS_CamDownload(LPCSTR pszCamFile, WORD wLoadAddress,
WORD wRunAddress, WORD& wNextLoadAddr, WORD& wNexRunAddr);
#else
BOOL TML_EXPORT TS_CamDownload(LPCSTR pszCamFile, WORD wLoadAddress,
WORD wRunAddress, WORD* wNextLoadAddr, WORD* wNexRunAddr);
#endif
/************************************************************************
 Function: Download a CAM file to the drive, at a specified address.
 Input arguments:
        pszCamFile: the name of the file containing the CAM information
        wLoadAddress: memory address where the CAM is loaded
        wRunAddress: memory where the actual CAM table is transfered and
executed at run time
 Output arguments:
        wNextLoadAddr: memory address available for the next CAM file;
if 0 there is no memory left
        wNextRunAddress: memory where the next CAM table is transfered
and executed at run time;
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


BOOL TML_EXPORT TS_CamInitialization(WORD LoadAddress, WORD RunAddress);
/************************************************************************
 Function:       Copies a CAM file from E2ROM to RAM memory. You should
not use this if you download CAMs directly to RAM memory
                                (load address == run address)
 Input arguments:
        LoadAddress: memory address in E2ROM where the CAM is already
loaded
         RunAddress: memory address in RAM where the CAM is copied.
 Output arguments:
        return:         TRUE if no error; FALSE if error
 ************************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_SetCammingSlaveRelative(WORD RunAddress, short
ReferenceBase, short EnableSlave, short MoveMoment,

                  long OffsetFromMaster = 0, double MultInputFactor = 0,
double MultOutputFactor = 0);
#else
BOOL TML_EXPORT TS_SetCammingSlaveRelative(WORD RunAddress, short
ReferenceBase, short EnableSlave, short MoveMoment,

                  long OffsetFromMaster, double MultInputFactor, double
MultOutputFactor);
#endif
/************************************************************************
 Function: Setup slave parameters in relative camming mode.
 Input arguments:
        RunAddress: memory addresses where the CAM is executed at run
time. If any of them is 0 it means that no start address is set
```

```
            ReferenceBase:
                    FROM_MEASURE -> the position reference starts from the
actual measured position value
                    FROM_REFERENCE -> the position reference starts from the
actual reference position value
            EnableSlave:
                    SLAVE_NONE -> do not enable slave operation
                    SLAVE_COMMUNICATION_CHANNEL -> enable operation got via
a communication channel
                    SLAVE_2ND_ENCODER -> enable operation read from 2nd
encoder or P&D inputs
            MoveMoment:
                    UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                    UPDATE_IMMEDIATE -> start moving immediate
                    UPDATE_ON_EVENT -> start moving on event
            nOffsetFromMaster, nMultInputFactor, nMultOutputFactor: if non-
zero, set the correspondent parameter
 Output arguments:
            return:          TRUE if no error; FALSE if error
*********************************************************************/

#ifdef __cplusplus
BOOL TML_EXPORT TS_SetCammingSlaveAbsolute(WORD RunAddress, double
LimitVariation, short ReferenceBase, short EnableSlave, short
MoveMoment,

                    long OffsetFromMaster = 0, double MultInputFactor = 0,
double MultOutputFactor = 0);
#else
BOOL TML_EXPORT TS_SetCammingSlaveAbsolute(WORD RunAddress, double
LimitVariation, short ReferenceBase, short EnableSlave, short
MoveMoment,

                    long OffsetFromMaster, double MultInputFactor, double
MultOutputFactor);
#endif
/*********************************************************************
 Function: Setup slave parameters in absolute camming mode.
 Input arguments:
            RunAddress: memory addresses where the CAM is executed at run
time. If any of them is 0 it means that no start address is set
            LimitVariation:
                    NO_VARIATION (0) -> no limitation on speed value at the
value set in the Drive Setup
                    A value which can be an acceleration or speed in
function of the reference type.
            ReferenceBase:
                    FROM_MEASURE -> the position reference starts from the
actual measured position value
                    FROM_REFERENCE -> the position reference starts from the
actual reference position value
```

```
        EnableSlave:
                SLAVE_NONE -> do not enable slave operation
                SLAVE_COMMUNICATION_CHANNEL -> enable operation got via
a communication channel
                SLAVE_2ND_ENCODER -> enable operation read from 2nd
encoder or P&D inputs
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
        nOffsetFromMaster, nMultInputFactor, nMultOutputFactor: if non-
zero, set the correspondent parameter
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************
/

BOOL TML_EXPORT TS_SetMasterResolution(long MasterResolution);
/*********************************************************************
 Function: Setup the resolution for the master encoder connected on the
second encoder input of the drive.
 Input arguments:
        MasterResolution:
                FULL_RANGE (0) -> select this option if the master
position is not cyclic. (e.g. the resolution is equal with the whole
                                                                32-bit
range of position)
                Value that reprezents the number of lines of the 2nd
master encoder
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************/

BOOL TML_EXPORT TS_SendSynchronization (long Period);
/*********************************************************************
 Function: Setup drives to send synchronization messages.
 Input arguments:
        Period: the time period between 2 sends
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************/

BOOL TML_EXPORT TS_Stop(void);
/*********************************************************************
 Function: Stop the motion.
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************/

BOOL TML_EXPORT TS_QuickStopDecelerationRate(double Deceleration);
/*********************************************************************
```

```
 Function: Set the deceleration rate used for QuickStop or SCurve
positioning profile.
 Input Argumernts:
         Deceleration: the value of the deceleration rate
 Output arguments:
        return:         TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SendPVTPoint(long Position, double Velocity, unsigned
int Time, short PVTCounter);
/********************************************************************
 Function: Sends a PVT point to the drive.
 Input arguments:
   Position:    drive position for the desired point
   Velocity:    desired velocity of the drive at the point
   Time:                       amount of time for the segment
   PVTCounter: integrity counter for current PVT point
 Output arguments:
        return:         TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SendPVTFirstPoint(long Position,double
Velocity,unsigned int Time, short PVTCounter,
                                                        short
PositionType, long InitialPosition, short MoveMoment, short
ReferenceBase);
/********************************************************************
 Function: Sends the first point from a series of PVT points and sets
the PVT motion mode.
 Input arguments:
   Position:    drive position for the desired point
   Velocity:    desired velocity of the drive at the point
   Time:                       amount of time for the segment
   PVTCounter: integrity counter for current PVT point
        PositionType: ABSOLUTE_POSITION or RELATIVE_POSITION
   InitialPosition:    drive initial position at the start of an
absolute PVT movement.
                                        It is taken into
consideration only if an absolute movement is requested
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
        ReferenceBase:
                FROM_MEASURE -> the position reference starts from the
actual measured position value
                FROM_REFERENCE -> the position reference starts from the
actual reference position value
 Output arguments:
        return:         TRUE if no error; FALSE if error
********************************************************************/
```

```
BOOL TML_EXPORT TS_PVTSetup(short ClearBuffer, short IntegrityChecking,
short ChangePVTCounter,
                                        short
AbsolutePositionSource, short ChangeLowLevel, short PVTCounterValue,
short LowLevelValue);
/********************************************************************
 Function: For PVT motion mode parametrization and setup.
 Input arguments:
   ClearBuffer: 0 -> nothing
               1 -> clears the PVT buffer
   IntegrityChecking:  0 -> PVT integrity counter checking is active
(default)
                      1 -> PVT integrity counter checking is inactive
   ChangePVTCounter:   0 -> nothing
                      1 -> drive internal PVT integrity counter is
changed with the value specified PVTCounterValue
   AbsolutePositionSource:    specifies the source for the initial
position in case the PVT motion mode will be absolute
                                0 -> initial position read from PVTPOS0
                                1 -> initial position read from current
value of target positio (TPOS)
   ChangeLowLevel:              0 -> nothing
                                1 -> the parameter for BufferLow signaling
is changed with the value specified LowLevelValue
   PVTCounterValue:    New value for the drive internal PVT integrity
counter
   LowLevelValue:      New value for the level of the BufferLow signal
 Output arguments:
       return:          TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SendPTPoint(long Position, unsigned int Time, short
PTCounter);
/********************************************************************
 Function: Sends a PT point to the drive.
 Input arguments:
   Position:    drive position for the desired point
   Time:                 amount of time for the segment
   PTCounter:   integrity counter for current PT point
 Output arguments:
       return:          TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SendPTFirstPoint(long Position, unsigned int Time,
short PTCounter,
                                        short
PositionType, long InitialPosition, short MoveMoment, short
ReferenceBase);
/********************************************************************
 Function: Sends the first point from a series of PT points and sets the
PT motion mode.
```

```
   Input arguments:
         Position:       drive position for the desired point
         Time:           amount of time for the segment
         PTCounter:      integrity counter for current PT point
         PositionType: ABSOLUTE_POSITION or RELATIVE_POSITION
         InitialPosition:        drive initial position at the start of an
absolute PT movement. It is taken into consideration only if an absolute
movement is requested
         MoveMoment:
                 UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                 UPDATE_IMMEDIATE -> start moving immediate
                 UPDATE_ON_EVENT -> start moving on event
         ReferenceBase:
                 FROM_MEASURE -> the position reference starts from the
actual measured position value
                 FROM_REFERENCE -> the position reference starts from the
actual reference position value
 Output arguments:
         return:         TRUE if no error; FALSE if error
**********************************************************************/
BOOL TML_EXPORT TS_PTSetup(short ClearBuffer, short IntegrityChecking,
short ChangePTCounter,
                                                      short
AbsolutePositionSource, short ChangeLowLevel, short PTCounterValue,
short LowLevelValue);
/**********************************************************************
 Function: For PT motion mode parametrization and setup.
 Input arguments:
   ClearBuffer:0 -> nothing
               1 -> clears the PT buffer
   IntegrityChecking:  0 -> PT integrity counter checking is active
(default)
                       1 -> PT integrity counter checking is inactive
   ChangePVTCounter:   0 -> nothing
                       1 -> drive internal PT integrity counter is
changed with the value specified PTCounterValue
   AbsolutePositionSource:    specifies the source for the initial
position in case the PT motion mode will be absolute
                          0 -> initial position read from PVTPOS0
                          1 -> initial position read from current value
of target positio (TPOS)
   ChangeLowLevel:     0 -> nothing
                       1 -> the parameter for BufferLow signaling is
changed with the value specified LowLevelValue
   PTCounterValue:     New value for the drive internal PT integrity
counter
   LowLevelValue:      New value for the level of the BufferLow signal
 Output arguments:
         return:         TRUE if no error; FALSE if error
**********************************************************************/
```

```
BOOL TML_EXPORT TS_MoveSCurveRelative(long RelPosition, double Speed,
double Acceleration, long JerkTime, short MoveMoment, short
DecelerationType);
/************************************************************************
 Function: For relative S-Curve motion mode.
 Input arguments:
        RelPosition:   Relative position reference value
        Speed:         Slew speed
        Acceleration:  Acceleration  decceleration
        JerkTime:      The time after the acceleration reaches the
desired value.
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
        DecelerationType:
                S_CURVE_SPEED_PROFILE -> s-curve speed profile
                TRAPEZOIDAL_SPEED_PROFILE -> trapezoidal speed profile
 Output arguments:
        return:        TRUE if no error; FALSE if error
*************************************************************************/


BOOL TML_EXPORT TS_MoveSCurveAbsolute(long AbsPosition, double Speed,
double Acceleration, long JerkTime, short MoveMoment, short
DecelerationType);
/************************************************************************
 Function: For absolute S-Curve motion mode.
 Input arguments:
        AbsPosition:   Absolute position reference value
        Speed:         Slew speed
        Acceleration:  Acceleration  decceleration
        JerkTime:      The time after wich the acceleration reaches the
desired value.
        MoveMoment:
                UPDATE_NONE -> setup motion parameters, movement will
start latter (on an Update command)
                UPDATE_IMMEDIATE -> start moving immediate
                UPDATE_ON_EVENT -> start moving on event
        DecelerationType:
                S_CURVE_SPEED_PROFILE -> s-curve speed profile
                TRAPEZOIDAL_SPEED_PROFILE -> trapezoidal speed profile
 Output arguments:
        return:        TRUE if no error; FALSE if error
*************************************************************************/


/***************************************************************/
/******************EVENT-RELATED functions**********************/
/***************************************************************/

#ifdef __cplusplus
BOOL TML_EXPORT TS_CheckEvent(BOOL& event);
```

```
#else
BOOL TML_EXPORT TS_CheckEvent(BOOL* event);
#endif
/***********************************************************************
 Function: Check if the actually active event occured.
 Output arguments:
 event: TRUE on event detected
        return:        TRUE if no error; FALSE if error
***********************************************************************/

BOOL TML_EXPORT TS_SetEventOnMotionComplete(BOOL WaitEvent, BOOL
EnableStop);
/***********************************************************************
 Function: Setup event when the motion is complete.
 Input arguments:
        WaitEvent:     TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:    TRUE -> On motion complete, stop the motion,
FALSE -> Don't stop the motion
 Output arguments:
        return:        TRUE if no error; FALSE if error
***********************************************************************/

BOOL TML_EXPORT TS_SetEventOnMotorPosition(short PositionType, long
Position, BOOL Over, BOOL WaitEvent, BOOL EnableStop);
/***********************************************************************
 Function: Setup event when motor position is over/under imposed value.
 Input arguments:
        PositionType: ABSOLUTE_POSITION or RELATIVE_POSITION
        Position:      Position value to be reached
        Over:          TRUE -> Look for position over; FALSE -> Look for
position below
        WaitEvent:     TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:    TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:        TRUE if no error; FALSE if error
***********************************************************************/

BOOL TML_EXPORT TS_SetEventOnLoadPosition(short PositionType, long
Position, BOOL Over, BOOL WaitEvent, BOOL EnableStop);
/***********************************************************************
 Function: Setup event when load position is over/under imposed value.
 Input arguments:
        PositionType: ABSOLUTE_POSITION or RELATIVE_POSITION
        Position:      Position value to be reached
        Over:          TRUE -> Look for position over; FALSE -> Look for
position below
        WaitEvent:     TRUE -> Wait until event occurs; FALSE ->
Continue
```

```
        EnableStop:      TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:          TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SetEventOnMotorSpeed(double Speed, BOOL Over, BOOL
WaitEvent, BOOL EnableStop);
/********************************************************************
 Function: Setup event when motor speed is over/under imposed value.
 Input arguments:
        Speed:           Speed value to be reached
        Over:            TRUE -> Look for speed over; FALSE -> Look for
speed below
        WaitEvent:       TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:      TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:          TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SetEventOnLoadSpeed(double Speed, BOOL Over, BOOL
WaitEvent, BOOL EnableStop);
/********************************************************************
 Function: Setup event when load speed is over/under imposed value.
 Input arguments:
        Speed:           Speed value to be reached
        Over:            TRUE -> Look for speed over; FALSE -> Look for
speed below
        WaitEvent:       TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:      TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:          TRUE if no error; FALSE if error
********************************************************************/


BOOL TML_EXPORT TS_SetEventOnTime(WORD Time, BOOL WaitEvent, BOOL
EnableStop);
/********************************************************************
 Function: Setup event after a time interval.
 Input arguments:
        Time:            Time after which the event will be set
        WaitEvent:       TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:      TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:          TRUE if no error; FALSE if error
********************************************************************/
```

```
BOOL TML_EXPORT TS_SetEventOnPositionRef(long Position, BOOL Over, BOOL
WaitEvent, BOOL EnableStop);
/***********************************************************************
 Function: Setup event when position reference is over/under imposed
value.
 Input arguments:
        Position:       Position value to be reached
        Over:           TRUE -> Look for speed over; FALSE -> Look for
speed below
        WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:     TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:         TRUE if no error; FALSE if error
***********************************************************************/

BOOL TML_EXPORT TS_SetEventOnSpeedRef(double Speed, BOOL Over, BOOL
WaitEvent, BOOL EnableStop);
/***********************************************************************
 Function: Setup event when speed reference is over/under imposed value.
 Input arguments:
        Speed:          Speed value to be reached
        Over:           TRUE -> Look for speed over; FALSE -> Look for
speed below
        WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:     TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:         TRUE if no error; FALSE if error
***********************************************************************/

BOOL TML_EXPORT TS_SetEventOnTorqueRef(int Torque, BOOL Over, BOOL
WaitEvent, BOOL EnableStop);
/***********************************************************************
 Function: Setup event when torque reference is over/under imposed
value.
 Input arguments:
        Torque:         Torque value to be reached
        Over:           TRUE -> Look for speed over; FALSE -> Look for
speed below
        WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
Continue
        EnableStop:     TRUE -> On event, stop the motion, FALSE -> Don't
stop the motion
 Output arguments:
        return:         TRUE if no error; FALSE if error
***********************************************************************/

BOOL TML_EXPORT TS_SetEventOnEncoderIndex(short IndexType, short
TransitionType, BOOL WaitEvent, BOOL EnableStop);
```

```
/*********************************************************************
 Function: Setup event when encoder index is triggered.
 Input arguments:
         IndexType:      INDEX_1 or INDEX_2
         TransitionType:        TRANSITION_HIGH_TO_LOW or
 TRANSITION_LOW_TO_HIGH
         WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
 Continue
         EnableStop:     TRUE -> On event, stop the motion, FALSE -> Don't
 stop the motion
 Output arguments:
         return:         TRUE if no error; FALSE if error
 *********************************************************************/

BOOL TML_EXPORT TS_SetEventOnLimitSwitch(short LSWType, short
TransitionType, BOOL WaitEvent, BOOL EnableStop);
/*********************************************************************
 Function: Setup event when selected limit switch is triggered.
 Input arguments:
         LSWType:        LSW_NEGATIVE or LSW_POSITIVE
         TransitionType:        TRANSITION_HIGH_TO_LOW or
 TRANSITION_LOW_TO_HIGH
         WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
 Continue
         EnableStop:     TRUE -> On event, stop the motion, FALSE -> Don't
 stop the motion
 Output arguments:
         return:         TRUE if no error; FALSE if error
 *********************************************************************/

BOOL TML_EXPORT TS_SetEventOnDigitalInput(BYTE InputPort, short IOState,
BOOL WaitEvent, BOOL EnableStop);
/*********************************************************************
 Function: Setup event when selected input port status is IOState.
 Input arguments:
         InputPort:      Input port number
         IOState:        IO_LOW or IO_HIGH
         WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
 Continue
         EnableStop:             TRUE -> On event, stop the motion, FALSE
 -> Don't stop the motion
 Output arguments:
         return:         TRUE if no error; FALSE if error
 *********************************************************************/

BOOL TML_EXPORT TS_SetEventOnHomeInput(short IOState, BOOL WaitEvent,
BOOL EnableStop);
/*********************************************************************
 Function: Setup event when selected input port status is IOState.
 Input arguments:
         IOState:        IO_LOW or IO_HIGH
```

```
           WaitEvent:      TRUE -> Wait until event occurs; FALSE ->
Continue
           EnableStop:           TRUE -> On event, stop the motion, FALSE
-> Don't stop the motion
 Output arguments:
           return:         TRUE if no error; FALSE if error
*********************************************************************/


/*******************************************************************/
/*******************INPUT / OUTPUT functions**********************/
/*******************************************************************/

BOOL TML_EXPORT TS_SetupInput(BYTE nIO);
/***********************************************************************
 Function: Setup IO port as input.
 Input arguments:
           nIO:    Port number to be set as input
 Output arguments:
           return:         TRUE if no error; FALSE if error
*********************************************************************/

#ifdef __cplusplus
BOOL TML_EXPORT TS_GetInput(BYTE nIO, BYTE& InValue);
#else
BOOL TML_EXPORT TS_GetInput(BYTE nIO, BYTE* InValue);
#endif
/***********************************************************************
 Function: Get input port status.
 Input arguments:
           nIO:    Input port number to be read
 Output arguments:
           InValue:        the input port status value (0 or 1)
           return:         TRUE if no error; FALSE if error
*********************************************************************/

#ifdef __cplusplus
BOOL TML_EXPORT TS_GetHomeInput(BYTE& InValue);
#else
BOOL TML_EXPORT TS_GetHomeInput(BYTE* InValue);
#endif
/***********************************************************************
 Function: Get home input port status.

 Output arguments:
           InValue:        the input port status value (0 or 1)
           return:         TRUE if no error; FALSE if error
*********************************************************************/

BOOL TML_EXPORT TS_SetupOutput(BYTE nIO);
/***********************************************************************
 Function: Setup IO port as output.
 Input arguments:
```

```
           nIO:     Port number to be set as output
 Output arguments:
           return:            TRUE if no error; FALSE if error
 ***********************************************************************/


BOOL TML_EXPORT TS_SetOutput(BYTE nIO, BYTE OutValue);
/***********************************************************************
 Function: Set output port status.
 Input arguments:
           nIO:     Output port number to be written
           OutValue:                  Output port status value to be set (0 or
1)
 Output arguments:
           return:            TRUE if no error; FALSE if error
 ***********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_GetMultipleInputs(LPCSTR pszVarName, short& Status);
#else
BOOL TML_EXPORT TS_GetMultipleInputs(LPCSTR pszVarName, short* Status);
#endif
/***********************************************************************
 Function:      Read multiple inputs.
 Input arguments:
           pszVarName: temporary variable name used to read input status
 Output arguments:
           Status: value of multiple input status.
           return:            TRUE if no error; FALSE if error
 ***********************************************************************/


BOOL TML_EXPORT TS_SetMultipleOutputs(LPCSTR pszVarName, short Status);
/***********************************************************************
 Function:      Set multiple outputs (for firmware versions FAxx).
           pszVarName: temporary variable name used to set output status
           Status: value to be set
 Output arguments:
           return:            TRUE if no error; FALSE if error
 ***********************************************************************/


BOOL TML_EXPORT TS_SetMultipleOutputs2(short SelectedPorts, short
Status);
/***********************************************************************
 Function:      Set multiple outputs (for firmware versions FBxx).
           SelectedPorts: port mask. Set bit n to 1 if you want to update
the status of port n.
           Status: value to be set
 Output arguments:
           return:            TRUE if no error; FALSE if error
 ***********************************************************************/


/****************************************************************/
/******************General use*****************************/
```

```
/******************************************************************/


BOOL TML_EXPORT TS_SendDataToHost(BYTE HostAddress, DWORD StatusRegMask,
WORD ErrorRegMask);
/**********************************************************************
 Function: Send status and error registers to host.
 Input arguments:
        HostAddress:   axis ID of host
        StatusRegMask: bit mask for status register
        ErrorRegMask: bit mask for error register
 Output arguments:
        return:        TRUE if no error; FALSE if error
**********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_OnlineChecksum(WORD startAddress, WORD endAddress,
WORD& checksum);
#else
BOOL TML_EXPORT TS_OnlineChecksum(WORD startAddress, WORD endAddress,
WORD* checksum);
#endif
/**********************************************************************
 Function: Get checksum of a memory range.
                startAddress:  start memory address
                endAddress: end memory address
 Output arguments:
                checksum: checksum (sum modulo 0xFFFF) of a memory range
returned by the active drive/motor
        return:        TRUE if no error; FALSE if error
**********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_DownloadProgram(LPCSTR pszOutFile, WORD&
wEntryPoint);
#else
BOOL TML_EXPORT TS_DownloadProgram(LPCSTR pszOutFile, WORD*
wEntryPoint);
#endif
/**********************************************************************
 Function: Download a COFF formatted file to the drive, and return the
entry point of that file.
 Input arguments:
        pszOutFile:    Name of the output TML object file
 Output arguments:
        wEntryPoint: the entry point (start address) of the downloaded
file
        return:        TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_GOTO(WORD address);
/**********************************************************************
 Function: Execute a GOTO instruction on the drive.
```

```
  Input arguments:
        address: program memory address of the instruction
  Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_GOTO_Label(LPCSTR pszLabel);
/**********************************************************************
  Function: Execute a GOTO instruction on the drive.
  Input arguments:
        pszLabel: label of the instruction
  Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_CALL(WORD address);
/**********************************************************************
  Function: Execute a CALL instruction on the drive.
  Input arguments:
        address: address of the procedure
  Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_CALL_Label(LPCSTR pszFunctionName);
/**********************************************************************
  Function: Execute a CALL instruction on the drive.
  Input arguments:
        pszFunctionName: name of the procedure to be executed
  Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_CancelableCALL(WORD address);
/**********************************************************************
  Function: Execute a cancelable call (CALLS) instruction on the drive.
  Input arguments:
        address: address of the procedure
  Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_CancelableCALL_Label(LPCSTR pszFunctionName);
/**********************************************************************
  Function: Execute a cancelable call (CALLS) instruction on the drive.
  Input arguments:
        pszFunctionName: name of the procedure to be executed
  Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_ABORT(void);
```

```
/************************************************************************
 Function:   Execute ABORT instruction on the drive (aborts execution of
a procedure called
                         with cancelable call instruction).
 Output arguments:
        return:          TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_Execute(LPCSTR pszCommands);
/************************************************************************
 Function: Execute TML commands entered in TML source code format (as is
entered in Command Interpreter).
 Input arguments:
        pszCommands: String containing the TML source code to be
executed. Multiple lines are allowed.
 Output arguments:
        return:          TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_ExecuteScript(LPCSTR pszFileName);
/************************************************************************
 Function: Execute TML commands in TML source code, from a script file
(as is entered in Command Interpreter).
 Input arguments:
        pszFileName: The name of the file containing the TML source code
to be executed.
 Output arguments:
        return:          TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_GetOutputOfExecute(LPSTR pszOutput, int nMaxChars);
/************************************************************************
 Function: Return the TML output code of the last previously executed
library function call.
 Input arguments:
        pszOutput: String containing the TML source code generated at
the last library function call.
        nMaxChars: maximum number of characters to return in the string
 Output arguments:
        return:          TRUE if no error; FALSE if error
 ************************************************************************/

BOOL TML_EXPORT TS_SetupLogger(WORD wLogBufferAddr, WORD wLogBufferLen,
WORD* arrayAddresses, WORD countAddr, WORD period);
/************************************************************************
 Function: Setup logger parameters (could be set up on a group/broadcast
destination).
 Input arguments:
        wLogBufferAddr: The address of logger buffer in drive memory,
where data will be stored during logging
        wLogBufferLen: The length in WORDs of the logger buffer
```

```
        arrayAddresses: An array containing the drive memory addresses
to be logged
        countAddr: The number of memory addresses to be logged
        period: How offen to log the data: a value between 1 and 7FFF
(useful only for new generation drives).
                If it is different than 1, one set of data will be
stored at every "period" samplings.
 Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


BOOL TML_EXPORT TS_StartLogger(WORD wLogBufferAddr, BYTE LogType);
/**********************************************************************
 Function: Start the logger on a drive (could be started on a
group/broadcast destination).
 Input arguments:
        wLogBufferAddr: address of logger buffer (previously set by
TS_SetupLogger)
        LogType:
                        LOGGER_FAST: logging occurs in fast sampling
control loop (current loop)
                        LOGGER_SLOW: logging occurs in slow sampling
control loop (position/speed loop)
 Output arguments:
        return:         TRUE if no error; FALSE if error
**********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_CheckLoggerStatus(WORD wLogBufferAddr, WORD& status);
#else
BOOL TML_EXPORT TS_CheckLoggerStatus(WORD wLogBufferAddr, WORD* status);
#endif
/**********************************************************************
 Function: Check logger status. (destination must be a single axis).
 Input arguments:
        wLogBufferAddr: address of logger buffer (previously set by
TS_SetupLogger)
 Output arguments:
        status: Number of points still remaining to capture; if it is 0,
the logging is completed
        return:         TRUE if no error; FALSE if error
**********************************************************************/


#ifdef __cplusplus
BOOL TML_EXPORT TS_UploadLoggerResults(WORD wLogBufferAddr, WORD*
arrayValues, WORD& countValues);
#else
BOOL TML_EXPORT TS_UploadLoggerResults(WORD wLogBufferAddr, WORD*
arrayValues, WORD* countValues);
#endif
/**********************************************************************
```

```
 Function: Upload logged data from the drive (destination must be a
single axis).
 Input arguments:
        wLogBufferAddr: address of logger buffer (previously set by
TS_SetupLogger)
        arrayValues:    Pointer to the array where the uploaded data is
stored on the PC
        countValues:    The size of arrayValues
 Output arguments:
        arrayValues:    uploaded logger data
        countValues:    The size of actualized data (lower or equal with
countValues input value)
        return:         TRUE if no error; FALSE if error
*********************************************************************/


void TML_EXPORT
TS_RegisterHandlerForUnrequestedDriveMessages(pfnCallbackRecvDriveMsg
handler);
/*********************************************************************
 Function: Register application's handler for unrequested drive
messages.
 Input arguments:
 pfnCallbackRecvDriveMsg:                pointer to handler
 Output arguments:
*********************************************************************/


BOOL TML_EXPORT TS_CheckForUnrequestedDriveMessages(void);
/*********************************************************************
 Function: Check if there are new unrequested drive messages and call
handler for every message received.
 Input arguments:
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************/


BOOL TML_EXPORT TS_DriveInitialisation(void);
/*********************************************************************
 Function: Execute ENDINIT command and verify if the setup table is
valid. This function
                must be called only after TS_LoadSetup & TS_SetupAxis
& TS_SelectAxis are called.
 Input arguments:
 Output arguments:
        return:         TRUE if no error; FALSE if error
*********************************************************************/
#ifdef __cplusplus
}
#endif

#endif //__TML_LIB_H__
```

TECHNOSOFT