Programming the ROBO TX Controller

Part 2: Windows Library "ftMscLib"

Library V1.5.11 dated 2/19/2012
(document version = library version)

References:

Description	Version	Date
ROBO TX Controller firmware	1.30	3/19/2012
ROBO Pro	3.1.3	3/26/2012
"PC_Programming_RoboTXC" package	1.5	4/24/2012

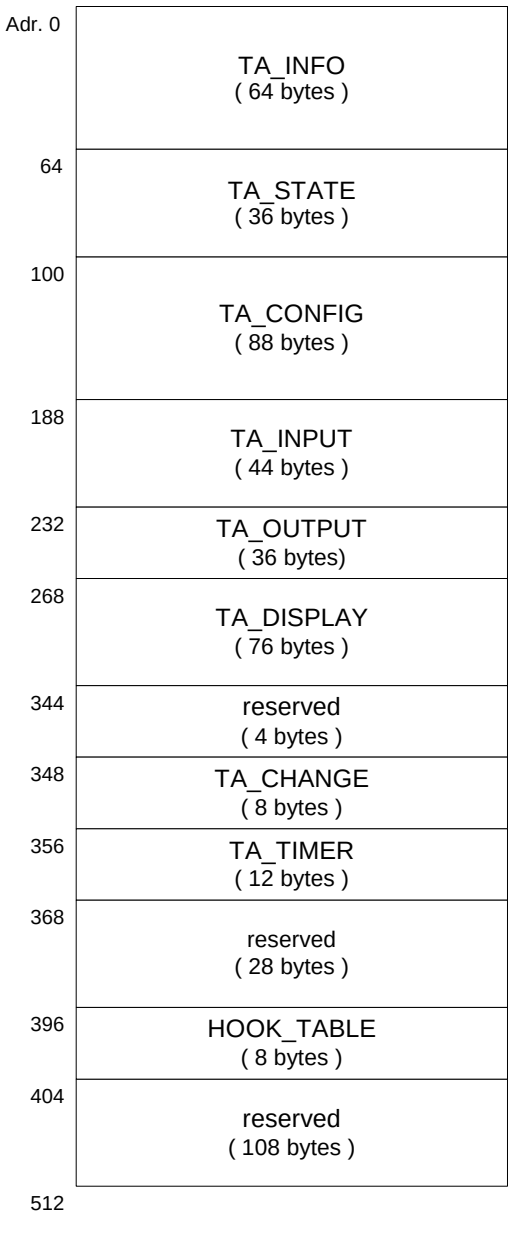
MSC Vertriebs GmbH
Design Center Aachen
Pascalstr. 21
52076 Aachen, Germany

Contents

1 General information - transfer area

As with the previous *FtLib* library by Knobloch and the previous *ROBO Interface*, what is called a transfer area is used to control the ROBO TX Controller. The output and input values are set or read out in this transfer area. Then a communication thread synchronizes the values with the ROBO TX Controller or updates the values received from the ROBO TX Controller in the transfer area. The ftMscLib library manages this transfer area as well as the configuration of the communication thread. As an API, the library provides a set of functions, each of which sets the output and configuration values in the transfer area or reads out the currently existing input values of the ROBO TX Controller.

Structure of the new transfer area in the ftMscLib and firmware:



Version V1.04.19 (2009/09/01)

Chapter 9 describes the memory layout structure in more detail.

To control larger models, it may be necessary to connect multiple ROBO TX Controllers. This is done via the integrated "extension ports" through a master/slave structure. In a chain of connected controllers, there is always one master and up to 8 slaves, which are controlled remotely by the master.

Each ROBO TX Controller has 2 extension ports that represent a serial expansion bus. A connection is made from one controller to the adjacent controller using a 6-pin ribbon cable. A data connection is thus made from the master to each of the slaves.

The expansion bus is an RS-485 interface with taps on each controller. The expansion bus is through-connected between the two connections on the controller, providing for simple wiring of adjacent controllers.

Communication on the expansion bus is also master/slave oriented. The master polls all connected slaves (extension controllers) cyclically and exchanges I/O information.

To do this, the memory of each controller contains a set of 9 transfer areas, a master and 8 slaves. Its own transfer area is always a master and is at the beginning of the occupied memory area. The slaves are controlled by the master in online mode in the same way as if controlling a single controller from a PC in online mode. The controller which is declared the master is then the only one that can communicate with the PC. Requests from the PC to a slave in online mode are routed by the master over the relevant transfer area and then over the expansion bus to the respective slave controller.

For many function calls, the result is that a unique controller ID must be provided so that the particular request reaches the right controller via the associated transfer area on the master controller.

To give a controller a unique address, the following definitions (controller IDs) need to have been set:
(from ROBO_TX_FW.h)

```
enum ta_id_e
{
    TA_LOCAL = 0,           // Transfer Area Master Controller
    TA_EXT_1,               // Transfer Area Extension 1 Controller
    TA_EXT_2,               // Transfer Area Extension 2 Controller
    TA_EXT_3,               // Transfer Area Extension 3 Controller
    TA_EXT_4,               // Transfer Area Extension 4 Controller
    TA_EXT_5,               // Transfer Area Extension 5 Controller
    TA_EXT_6,               // Transfer Area Extension 6 Controller
    TA_EXT_7,               // Transfer Area Extension 7 Controller
    TA_EXT_8,               // Transfer Area Extension 8 Controller
    TA_COUNT                // Number of Transfer Areas in array = 9
};
```

2 General functions of the "ftMscLib" library

2.1 ftxGetLibVersion

DWORD ftxGetLibVersion (void)

Version number of the current library as DWORD value (4 byte), format: 3.2.1.0

Byte 3: 0
Byte 2: Main version
Byte 1: Release version
Byte 0: Sub-version

2.2 ftxGetLibVersionStr

DWORD ftxGetLibVersionStr (LPSTR strBuff,
DWORD maxLen)

The version number of the library is returned as a string.
Format "MM.RR.SS yyyy/mm/dd"
MM = main version, RR = release version, SS = sub-version, yyyy/mm/dd = date

Call:	LPSTR strBuff	- string buffer pointer for receiving the version string
	DWORD maxLen	- reserved length of string buffer
Return:	DWORD len	- length of version string or 0

2.3 ftxInitLib

DWORD ftxInitLib (void)

Library initialization function. To use the library functionality, this function must first be called. Logging is prepared for this; the internal variables and the FishX1 transfer area are initialized.

Return:	DWORD errCode	- error code (ftErrCode.h)
	FTLIB_ERR_SUCCESS	- no error
	FTLIB_ERR_LIB_IS_INITIALIZED	- library is already initialized
	FTLIB_ERR_NO_MEMORY	- no memory

2.4 ftxCloseLib

DWORD ftxCloseLib (void)

Counterpart to InitFtLib() function; frees up the reserved memory again. Closes the library.

Return:	DWORD errCode	- error code (ftErrCode.h)
	FTLIB_ERR_SUCCESS	- no error

2.5 ftxIsLibInit

DWORD ftxIsLibInit (void)

Provides information on whether the ftMscLib library is initialized.

Return:	DWORD errCode	- error code (ftErrCode.h)
	FTLIB_ERR_SUCCESS	- no error
	FTLIB_ERR_LIB_IS_INITIALIZED	- library is initialized
	FTLIB_ERR_LIB_IS_NOT_INITIALIZED	- library is not initialized

2.6 ftxOpenComDeviceNr

HANDLE ftxOpenComDeviceNr (DWORD port,
DWORD baudr,
*DWORD *errcode)*

The function opens for communication with the ROBO TX Controller with a COM interface specified with a number and returns a unique handle. Possible values for a COM interface: 1 to 255. The available port numbers come from the list in the Device Manager. The *errcode* variable is used for receiving a possible error code.

Call:	DWORD port	- port number of COM interface, e.g. 12 for COM12
	DWORD baudr	- baud rate, current 38400 fixed
	DWORD *errcode	- pointer to an error variable
Return:	HANDLE fthdl	- handle for communication with the ROBO TX Controller; if an error occurs (=NULL), then the *errcode* variable contains a possible error code.

Possible error code (ftErrCode.h)

FTLIB_ERR_SUCCESS	- no error (handle != NULL)
FTLIB_ERR_DEVICE_IS_OPEN	- COM is already open
FTLIB_ERR_SOME_DEVICES_ARE_OPEN	- a different COM is open
FTLIB_ERR_OPEN_COM	- error opening a COM

2.7 ftxOpenComDevice

*HANDLE ftxOpenComDevice (char *comStr,*
DWORD baudr,
*DWORD*errcode)*

Function equivalent to function from 2.6, but the particular COM interface can be passed as a string, e.g. "COM4" or "COM32".

Call:	char comStr	- COM interface with port number as string, e.g. "COM4"
	DWORD baudr	- baud rate, current 38400 fixed
	DWORD *errcode	- pointer to an error variable

Return: see 2.6

2.8 ftxCloseDevice

DWORD ftxCloseDevice (HANDLE fthdl)

The function closes an open connection to the specified ROBO TX Controller. All active threads for communication with the ROBO TX Controller are stopped.

Call:	HANDLE fthdl	- valid handle of a COM interface
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

2.9 ftxCloseAllDevices

DWORD ftxCloseAllDevices (void)

Function closes all open connections to all possible ROBO TX Controllers. Currently supports only one connection to a ROBO TX Controller. When the function is called, the only existing connection is closed.

Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

2.10 ftxIsHandleValid

DWORD ftxIsHandleValid (HANDLE fthdl)

Checks whether the specified handle of a COM interface is (still) valid.

Call: HANDLE fthdl - valid handle of a COM interface

Return: DWORD errCode - error code (ftErrCode.h)

FTLIB_ERR_SUCCESS	- handle is valid, no error
FTLIB_ERR_DEVICE_NOT_OPEN	- no COM is open
FTLIB_ERR_UNKNOWN_DEVICE_HANDLE	- the specified handle is invalid or unknown

2.11 GetComStatus

DWORD GetComStatus (HANDLE fthdl)

Same functionality as the function "ftxIsHandleValid()"; see 2.10.

2.12 GetAvailableComPorts

DWORD GetAvailableComPorts (int selectMode)

Function finds all available COM ports based on information in the Windows Registry and returns the number of ports. This information is managed internally in a static list for further use. To find and display the available COM ports, this function must be called first.

Call: int selectMode - information on whether all COM ports are read or only the ports that are intended for connection with a ROBO TX Controller (USB or Bluetooth)

ALL_PORTS	0	all ports
USB_ONLY	1	USB ports only
BT_ONLY	2	Bluetooth connections only

Return: DWORD iComPorts - number of COM ports found in the internal list

2.13 EnumComPorts

DWORD EnumComPorts (DWORD idx,
LPSTR comstr,
DWORD maxlen)

Provides an entry from the static list created by the library which contains the available COM ports. The function "GetAvailableComPorts()" must have been called previously. The entry is returned as a string with information on the COM port and a description of its use, as taken from the Windows Registry, e.g.: "COM32 (fischertechnik USB ROBO TX Controller)". The passed string buffer must be dimensioned accordingly so that it can receive the string with the description.

Call:	DWORD idx	- list index within the list of available COM ports 0 < idx < iComPorts (GetAvailableComPorts())
	LPSTR comstr	- string buffer for receiving the COM description
	DWORD maxLen	- max. length of string buffer
Return:	DWORD index	- index of the list entry which was read (= idx); in the event of an error, the value FTLIB_ERR_INVALID_PARAM is returned

2.14 ftxGetLibErrorString

DWORD ftxGetLibErrorString (DWORD errCode,
DWORD typ,
LPSTR strBuff,
DWORD maxLen)

Function returns either a representative error message using the passed error code or the error constant itself as a string. No error analysis can be made as a result, even if the error code is unknown.

Call:	DWORD errCode	- error code
	DWORD typ	- type of the returned string 0 = error constant as string 1 = error text in English
	LPSTR strBuff	- buffer for receiving the error message (null-term. string)
	DWORD maxLen	- length of the recording buffer
Return:	DWORD len	- length of the copied error message or 0

2.15 ftxGetManufacturerStrg

DWORD ftxGetManufacturerStrg (HANDLE fthdl,
LPSTR strBuff,
DWORD maxlen)

The function returns a null-terminated string with information on the manufacturer, currently still static "MSC Vertriebs GmbH".

Note: This information is currently not yet read out of the ROBO TX Controller.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	LPSTR strBuff	- string buffer for receiving the manufacturer information
	DWORD maxlen	- available length of *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.16 ftxGetShortNameStrg

WORD ftxGetShortNameStrg (HANDLE fthdl,
LPSTR strBuff,
DWORD maxlen)

The function provides a null-terminated string with the official description "ROBO TX Controller".

Note: This information is currently not yet read out of the ROBO TX Controller.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	LPSTR strBuff	- string buffer for receiving the string of characters
	DWORD maxlen	- available length of *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.17 ftxGetLongNameStrg

DWORD ftxGetLongNameStrg (HANDLE fthdl,
LPSTR strBuff,
DWORD maxlen)

Calls the function "GetFtShortNameStrg()"; for parameter description, see 2.16

2.18 ftxGetFirmwareStrg

DWORD ftxGetFirmwareStrg (HANDLE fthdl,
LPSTR strBuff,
DWORD maxlen)

The function returns a null-terminated string with the firmware installed on the ROBO TX Controller. Format: 'V xx.yy' .

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	LPSTR strBuff	- string buffer for receiving the string of characters
	DWORD maxlen	- available length of *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.19 ftxGetSerialNrStrg

WORD ftxGetSerialNrStrg (HANDLE fthdl,
LPSTR strBuff,
DWORD maxlen)

The function returns a null-terminated string with the serial number of the connected ROBO TX Controller.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	LPSTR strBuff	- string buffer for receiving the string of characters
	DWORD maxlen	- available length of *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.20 GetRoboTxDevName

DWORD GetRoboTxDevName (HANDLE fthdl,
int devId,
LPSTR strBuff,
DWORD len)

The function returns a null-terminated string with the name of the specified ROBO TX Controller. The function copies the result into the *strBuff* string buffer, which must be dimensioned accordingly. The max. length of the name is 16 characters. By default, the name of the ROBO TX Controller is made up of the string "ROBO TX-" and a 3-digit number "xxx". The number assigned is the checksum of the given Bluetooth address as a decimal number.

Example: From the existing Bluetooth address of the controller, "00:13:7B:B2:16" the result is the checksum "1A8" (hex), which corresponds to "424" (decimal). The name of the present ROBO TX Controller is therefore "ROBO TX-424", which appears by default on the display of the controller.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer for receiving the controller name
	DWORD len	- max. length of passed string buffer *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.21 SetRoboTxDevName

DWORD SetRoboTxDevName (HANDLE fthdl,
int devId,
LPSTR strBuff)

Use this function to change the default name given to a ROBO TX Controller. The max. length of the host name is 16 characters.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer that contains the new name of the controller
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

2.22 GetRoboTxBtAddr

WORD GetRoboTxBtAddr (HANDLE fthdl,
int devId,
LPSTR strBuff,
DWORD len)

The function returns a null-terminated string with the given Bluetooth address of the connected ROBO TX Controller. The maximum length of the string for receiving the Bluetooth address is 17 characters.

Example: 00:13:7B:51:BF:8D

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer for receiving the Bluetooth address
	DWORD len	- max. length of passed string buffer *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.23 GetRoboTxFwStr

DWORD GetRoboTxFwStr (HANDLE fthdl,
int devId,
LPSTR strBuff,
DWORD len)

The function returns a null-terminated string with the version of the firmware currently installed on the ROBO TX Controller. Format: 'V xx.yy' .

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer for receiving the firmware version
	DWORD len	- max. length of passed string buffer *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.24 GetRoboTxFwVal

DWORD GetRoboTxFwVal (HANDLE fthdl,
int devId,
*DWORD *version)*

This function returns the firmware version as DWORD. This makes it possible to compare firmware versions. The value is composed of the version of the ROBOLIB.dll library (xx) and the version of the firmware (yy.zz):

	00	xx	yy	zz
Byte	4	3	2	1

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	DWORD *version	- variable for receiving a value for firmware
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

2.25 GetRoboTxHwStr

WORD GetRoboTxHwStr (HANDLE fthdl,
int devId,
LPSTR strBuff,
DWORD len)

This function returns a null-terminated string with the existing hardware release version.

Example: 'C'

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller

	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer for receiving the hardware release
	DWORD len	- max. length of passed string buffer *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.26 GetRoboTxSerialStr

DWORD GetRoboTxSerialStr (HANDLE fthdl,
int devId,
LPSTR strBuff,
DWORD len)

The function returns a null-terminated string with the serial number of the connected ROBO TX Controller.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer for receiving the hardware release
	DWORD len	- max. length of passed string buffer *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

2.27 GetRoboTxDllStr

DWORD GetRoboTxDllStr (HANDLE fthdl,
int devId,
LPSTR strBuff,
DWORD len)

The function returns a null-terminated string with the version of the "ROBOLIB.DLL" ROBO-TX library stored on the controller. Format: 'xx' .

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPSTR strBuff	- string buffer for receiving the hardware release
	DWORD len	- max. length of passed string buffer *strBuff*
Return:	DWORD len	- length of the copied string of characters, if error = 0

3 Online mode functions

3.1 ftxStartTransferArea

DWORD ftxStartTransferArea (HANDLE fthdl)

Function activates the transfer area in the library for online mode. The communication thread is started and carries out the IO commands in "online mode". It reads the current values from the output structure of the transfer area (configuration and output values) and sends these to the ROBO TX Controller. As a response to an IO request, the controller sends the current values and the communication thread then updates these values in the input structure of the transfer area.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error, thread is activated) or error code

3.2 ftxStopTransferArea

DWORD ftxStopTransferArea (HANDLE fthdl)

Function disables transfer area communication with the ROBO TX Controller. The communication thread is stopped. Communication with the ROBO TX Controller is stopped.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error, thread stopped) or error code

3.3 ftxIsTransferActiv

DWORD ftxIsTransferActiv (HANDLE fthdl)

Functions checks if the transfer area is active, i.e. if there is cyclical communication with the ROBO TX Controller.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
Return:	DWORD errCode	- error code (ftErrCode.h)
FTLIB_ERR_THREAD_NOT_RUNNING		- transfer thread is inactive
FTLIB_ERR_THREAD_SYNCHRONIZED		- thread synchronizes with the ROBO TX-C
FTLIB_ERR_THREAD_IS_RUNNING		- thread is active in online mode

3.4 GetTransferAreasArrayAddr

*volatile TA_ARRAY *GetTransferAreasArrayAddr (HANDLE fthdl)*

The function returns a pointer to the memory area of all transfer areas (transfer areas are arranged as an array of structures).

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
Return:	TA_ARRAY * adr	- starting address of the memory area of all transfer areas

All transfer areas (master + up to 8 slaves) are stored in this memory area. The structure of the transfer area is described in chapter 1. For the layout of the structures within the transfer area, refer to the header file ROBO_TX_FW.h.

3.5 GetTransferAreaStatusAddr

*TA_STATUS *GetTransferAreaStatusAddr (HANDLE fthdl, int devId)*

The function returns a pointer to the structure TA_STATUS within the transfer area. From this, the current status of the transfer area can be read.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
Return:	TA_STATUS * adr	- pointer to the TA_STATUS structure. The structure is defined in the ROBO_TX_FW.h header file.

Possible status messages:

TA_STATUS_STOP	0	- transfer area is not running
TA_STATUS_RUN	1	- transfer area is running
TA_STATUS_SYNC	2	- transfer area is being synchronized

3.6 StartCounterReset

DWORD StartCounterReset (HANDLE fthdl,
int devId,
int cntId)

Function starts the procedure for resetting the counter input on the ROBO TX Controller to 0. This is an asynchronous process and is therefore not completed directly on return of the function call. A confirmation may take place via callback, if required. To do this, a callback function must be set using SetCBCounterResetted (see below).

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int shmId	- controller ID (master or extension controller)
	int cntId	- counter index (0 to 3) of counter 1 to 4
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.7 SetCBCounterResetted

*void SetCBCounterResetted (void (__stdcall *) cbFunct (DWORD devId, DWORD cntId))*

Function installs the specified callback function in the library. The callback function reports the status "Counter input reset".

Callback function parameter:

Call:	DWORD devId	- controller ID (master or extension controller)
	DWORD cntId	- counter index (0 to 3) of counter 1 to 4

3.8 SetOutMotorValues

DWORD SetOutMotorValues (HANDLE fthdl,
int devId,
int motorId,
int duty_p,
int duty_m)

Function sets the duty values for the two motor outputs M+ and M- for a motor in the transfer area.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int motorId	- index of the motor to be controlled (0 to 3)
	int duty_p	- duty value for motor output M+
	int duty_m	- duty value for motor output M-
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.9 SetOutPwmValues

DWORD SetOutPwmValues (HANDLE fthdl,
int devId,
int outId,
int duty)

Function sets the specified duty value in the transfer area for a PWM output.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int outId	- index of the PWM output (0 to -7)
	int duty	- duty value for the PMW output
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.10 SetFtUniConfig

DWORD SetFtUniConfig (HANDLE fthdl,
int devId,
int ioId,
int mode,
BOOL digital)

Function configures a universal input (combination input) to measure analog and digital voltage and resistance values and for analog distance measuring.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int ioId	- index of the universal input (0 to 7)
	int mode	- measurement mode 0= voltage (mV), 1= resistance (5 kΩ) 3= ultrasonic distance sensor (distance measurement)
	BOOL digital	- identifier for whether the value is returned digitally (FALSE= analog, TRUE= digital), if it is a distance measurement, then only analog
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.11 SetFtCntConfig

DWORD SetFtCntConfig (HANDLE fthdl,
int devId,
int cntId,
int mode)

Function configures a counter input (counter); how the status of the counter is to be interpreted.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int cntId	- counter index (0 to 3)
	int mode	- 0= NORMAL mode, 1= INVERTED mode
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.12 SetFtMotorConfig

WORD SetFtMotorConfig (HANDLE fthdl,
int devId,
int motorId,
BOOL status)

Function activates or deactivates motor outputs. Analogous to that, the PMW outputs are deactivated or activated accordingly.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int motorId	- motor index (0 to 3)
	BOOL status	- TRUE= motor on (motor outputs activated)
		FALSE= motor off (PMW output activated)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.13 StartMotorExCmd

DWORD StartMotorExCmd (HANDLE fthdl,
int devId,
int mIdx,
int duty,
int mDirection,
int sIdx,
int sDirection,
int pulseCnt)

Function activates the intelligent motor mode for motor synchronization. The motor moves to the desired position using the shared counter information. The application shares the information that the motor has reached the end position by using a previously installed callback function (see also SetCBMotorExReached()).

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int mIdx	- motor index (0 to 3) from master (motor)
	int duty	- duty value for master/slave motor
	int mDirection	- direction for master motor (0= CW, 1= CCW)
	int sIdx	- motor index (0 to 3) from slave (motor)
	int sDirection	- direction for slave motor (0= CW, 1= CCW)
	int pulseCnt	- number of count pulses for moving to a position, relative to the starting position
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.14 StopMotorExCmd

DWORD StopMotorExCmd (HANDLE fthdl,
int devId,
int mtrIdx)

Function immediately stops the previously activated intelligent motor mode for motor synchronization of the specified motor by resetting the duty values as well as the distance value in the output structure to 0.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int mtrIdx	- motor index (0 to 3)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.15 StopAllMotorExCmd

DWORD StopAllMotorExCmd (HANDLE fthdl,
int devId)

Function immediately stops the previously activated intelligent motor mode for motor synchronization of all motors of the ROBO TX Controller by resetting the duty values as well as the distance values in the output structure to 0.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.16 SetCBMotorExReached

*void SetCBMotorExReached (void (__stdcall *) cbFunct (DWORD devId, DWORD mtrIdx))*

Function installs to the library the specified callback function that reports the "Motor Reached State" status during active motor synchronization (intelligent motor mode).

Callback function parameter:

Call:	DWORD devId	- controller ID (master or extension controller)
	DWORD mtrIdx	- motor index (0 to 3)

3.17 GetInIOValue

DWORD GetInIOValue (HANDLE fthdl,
int devId,
int ioId,
*INT16 *value,*
*BOOL32 *overrun)*

Function reads the current value of a universal input from the transfer area and makes it available to an application. The values of the universal inputs come as a response to an IO request from the ROBO TX Controller and are updated in the transfer area.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int ioId	- index, universal input (0 to 7)
	INT16 *value	- pointer to variable that receives the value
	BOOL32 *overrun	- pointer to a variable for overrun message FALSE: no overrun TRUE: overrun
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.18 GetInCounterValue

DWORD GetInCounterValue (HANDLE fthdl,
int devId,
int cntId,
*INT16 *count,*
*INT16 *state)*

Function reads the current value of a counter input (counter) from the transfer area and makes it available to an application.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	int cntId	- index, counter (0 to 3)
	INT16 *count	- pointer to variable, current counter status
	INT16 *state	- pointer to variable, set mode of counter TRUE: "INVERTED" mode FALSE: "NORMAL" mode
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.19 GetInDisplayButtonValue

DWORD GetInDisplayButtonValue (HANDLE fthdl,
int devId,
*INT16 *left,*
*INT16 *right)*

Function reads the current status of the two push-button switches on the display. The time for how long a push-button switch remains depressed is displayed.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	INT16 *left	- pointer to INT16 variable, push-button switch (left)
	INT16 *right	- pointer to INT16 variable, push-button switch (right)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.20 FtRemoteCmd

DWORD FtRemoteCmd (HANDLE fthdl,
*char *ftCmd,*
*void (__stdcall *) cbFunc (LPSTR strBuff, DWORD len))*

The function sends a console command to the ROBO TX Controller. The response from the ROBO TX Controller is passed to the callback function as a string buffer and can be analyzed accordingly. The language of the received data is also provided.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	char *ftCmd	- pointer to a null-terminated string with a (remote) console command
	cbFunc	- function pointer of a callback function for indicating or displaying the received remote data from the ROBO TX Controller
	LPSTR strBuff	- string buffer pointer with remote data
	DWORD len	- length of remote data
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.21 RTxCleanDisk

DWORD RTxCleanDisk (HANDLE fthdl,
DWORD device)

Function cleans the specified disk on the ROBO TX Controller. Cleaning of the disk is supported only for the RAMDISK and FLASH disks. Deleting all entries on the SYSTEM partition is not supported. On a FLASH disk, the file "sys_par.ini" will remain on the disk even after cleaning the disk.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	DWORD device	- target disk (RAM disk=0, flash=1)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.22 GetRoboTxMemLayout

DWORD GetRoboTxMemLayout (HANDLE fthdl,
int devId,
PULONG pTAarray,
PULONG pAppStart,
PULONG pAppSize)

Function reads the required address for a ROBO Pro program from the ROBO TX Controller. All addresses are within the address space of the CPU of the ROBO TX Controller (not in the PC address space).

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	PULONG pTAarray	- start address of the transfer area array memory area
	PULONG pAppStart	- application area start address
	PULONG pAppSize	- application area length
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

3.23 SetCBRoboExtState

*void SetCBRoboExtState (void (__stdcall *) cbFunc (DWORD devId, DWORD state))*

Function installs a callback function in the library which reports status messages from external ROBO TX Controllers that are in multi-controller mode (master/slave mode). The callback function is used to report whether a slave controller is online or offline or if it is or is not connected with the master.

Call:	cbFunc	- function pointer of a callback function
	DWORD devId	- controller ID of a slave controller
	DWORD state	- status as to whether SLAVE_OFFLINE=0 or SLAVE_ONLINE=1

3.24 SetRoboTxMessage

DWORD SetRoboTxMessage (HANDLE fthdl,
int devId,
LPCSTR msg)

Function generates a viewable text on the display of the ROBO TX Controller. The text can only be shown on the display when the transfer area is activated. The maximum text length is 98 characters (ASCII). When passing an empty text (= 0), the text currently shown on the display is deleted and the standard output of the ROBO TX Controller reappears. Alternatively, the messages displayed and buffered can be deleted on the controller using the two push-button switches.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
	LPCSTR msg	- text that is shown on the display; max. 98 characters
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

4 Functions for uploading data

4.1 FtFileUpload

DWORD FtFileUpload (HANDLE fthdl,
*char *fname,*
DWORD device,
*void (__stdcall *) cbFunc (DWORD status))*

Function carries out an upload of the specified file to the ROBO TX Controller. The file is specified using the complete path. The callback function to be specified provides information on the status, or the status of the transfer.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	char *fname	- pointer to a null-terminated string with the complete path name of the file or program.
	DWORD device	- target disk (RAM disk=0, flash=1, system=2)
	cbFunc	- function pointer of a callback function

Return: DWORD errCode - FTLIB_ERR_SUCCESS (no error) thread that executes the upload is activated, or error code

Possible status codes for the callback function:

`FTLIB_ERR_UPLOAD_START`	- upload started
`FTLIB_ERR_UPLOAD_TIMEOUT`	- upload ends with a timeout
`FTLIB_ERR_UPLOAD_CANCELED`	- upload canceled by remote party
`FTLIB_ERR_UPLOAD_FAILED`	- upload canceled due to an error
`FTLIB_ERR_UPLOAD_FILE_READ_ERR`	- error reading the file; upload canceled
`FTLIB_ERR_UPLOAD_NAK`	- NAK identifier (0x15) received Packet repeated (up to 5 attempts)
`FTLIB_ERR_UPLOAD_ACK`	- ACK identifier (0x06) received, next packet sent
`FTLIB_ERR_UPLOAD_DONE`	- upload completed successfully
`FTLIB_ERR_UPLOAD_FLASHWRITE`	- flash is written

4.2 FtRamFileUpload

DWORD FtRamFileUpload (HANDLE fthdl,
DWORD device,
CONST PVOID pProg,
DWORD size,
LPCSTR progname,
*void (__stdcall *) cbFunc (DWORD status))*

Function carries out an upload of a program or file to the ROBO TX Controller. A pointer to a memory area containing the data to be transmitted and the length of the bytes to be transmitted are specified. On the ROBO TX Controller, the file or the program is stored under the name given to it on the corresponding disk when called. The callback function provides information on the status, or the status during the transfer.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	DWORD device	- target disk (RAM disk=0, flash=1, system=2)
	CONST PVOID pProg	- pointer to the program for upload
	DWORD size	- program length to be transmitted
	LPSTR progname	- null-terminated string with the program name
	cbFunc	- function pointer of a callback function
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) thread that executes the upload is activated, or error code

For possible status codes of callback function, see 4.1.

4.3 RoboTxFwUpdate

DWORD RoboTxFwUpdate (HANDLE fthdl,
LPCSTR path,
*void (__stdcall *) cbFunc (DWORD status, LPSTR infoStr))*

The function generates a thread in the library which carries out a complete firmware update on the ROBO TX Controller. The files needed for the firmware update are in the specified directory. The application to be called shares information related to the course of actions via the installed callback function.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	LPCSTR path	- directory containing the firmware update files
	cbFunc	- function pointer of a callback function
	DWORD status	- firmware update status For definitions, see 6 below (error codes)
	LPSTR infoStr	- string with information on the executed action during the firmware update
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error) or error code

5 Program control functions

5.1 FtProgramRun

DWORD FtProgramRun (HANDLE fthdl,
int devId)

Function starts a loaded program. The program can be loaded e.g. via the function FtFileUpload() in the ROBO TX Controller. Then the stored program can be launched using this function.

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error, program started) or error code

5.2 FtProgramStop

DWORD FtProgramStop (HANDLE fthdl,
int devId)

Function stops and quits the running program on the ROBO TX Controller. The program can then be restarted using the function FtProgramRun().

Call:	HANDLE fthdl	- current handle of the ROBO TX Controller
	int devId	- controller ID (master or extension controller)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (no error, program stopped) or error code

6 Bluetooth Message API functions

A ROBO TX Controller can establish program-controlled Bluetooth connections to other ROBO TX Controllers and use this to send short text messages (e.g. 10 bytes). Programs can then synchronize different ROBO TX Controllers, for instance, remotely.

Additional information (e.g. the current remote signal strength) can also be displayed, and reactions can be made to remote events in particular (e.g. a dropped connection because the robot is moving out of range).

To use the Bluetooth messaging functions, at least two ROBO TX Controllers are required. For a test environment basic setup, it is recommended that you read the introductory chapter 6 covering "Bluetooth Messaging API" in the document "PC_Programming_RoboTXC.pdf" version 1.5.

6.1 Introduction

A maximum of 8 connections are managed in the library. Each connection is referenced via what is called a channel index which is specified by the application in a particular value range. When establishing a Bluetooth connection, the Bluetooth address is linked with the channel index to be specified once the connection is successfully established. Further accesses to a connection are then only made through the channel index. The application (in this case the C program) alone manages the channel index <–> Bluetooth address assignment.

Each function call with an access to a Bluetooth connection has a callback function as a parameter. The library manages these callback functions for each channel index and calls these as necessary. It is always the callback function specified last for a connection that is managed for the particular function.

6.2 On-call duty and receive ready status

The ROBO TX Controller does not automatically accept Bluetooth connections, but only accepts them once a program is launched and the program signals its on-call duty status using the function BtListenConOn(). If no program has been launched or the program is written in such a way that the on-call duty is not present at the start (but, for instance, conditionally at a later time), then the attempt by a remote party to connect is rejected with the status code 16 (BT_NO_LISTEN_ACTIVE).

The same behavior applies with the ready receive status of messages. In the case of an existing Bluetooth connection, only after the ready receive status has been activated by calling the function BtReadMsgOn() are the received messages reported. Otherwise, the received messages are dismissed in the firmware.

6.3 Loopback function

The loopback function is available for testing the Bluetooth Message API locally on a single ROBO TX Controller. In this case there is no remote connection between two controllers. Instead, this mode is only simulated locally.

When scanning using StartScanBtDevice(), the local controller's own Bluetooth address is always used first. If a connection is established to the controller's Bluetooth address using ConnectBtAddress(), a (simulated) Bluetooth connection is established immediately. The same applies when activating on-call duty on the controller's own Bluetooth address using BtListenConOn(). The channel number assigned to the controller's own Bluetooth address can be composed of any value from 1 to 8.

If messages are sent on a loopback connection using SendBtMessage(), these messages are returned locally to the firmware of the controller as an echo and received again immediately on the same connection (channel). However, in loopback mode it is also required that the ready receive status has been activated in advance by calling the function BtReadMsgOn().

6.4 StartScanBtDevice

DWORD StartScanBtDevice (HANDLE fthdl,
*void (__stdcall *) cbFunc (BT_SCAN_STATUS *result))*

Scans for Bluetooth devices in the vicinity of the ROBO TX Controller currently connected to the PC (not to be confused with searching around the PC). This command is used to start scanning for available Bluetooth devices. The called callback function then provides successive scanning results (Bluetooth devices found) and can be called multiple times. The entire scanning process up until the status BT_INQUIRY_SCAN_END is reported lasts approximately 30 seconds. By calling the function CancelScanBtDevice(), the process can also be canceled early.

Call parameters:

fthdl - valid handle of the local ROBO TX Controller
cbFunc - callback function that reports the results of the application scan (list of Bluetooth devices found)

Return code:

errCode - FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

result - pointer to data structure BT_SCAN_STATUS e.g. with a Bluetooth device found (status=BT_INQUIRY_SCAN_RESULT)

Data structure:

```
typedef struct {
      UINT16      status;                      // Search status
      UCHAR8      btaddr[6];                   // Bluetooth address
      BYTE        dummy_1[2];
      Char        devname[HOSTNAME_LEN + 1];   // Device name
      BYTE        dummy_2[3];
} BT_SCAN_STATUS;
```

Possible status codes after the scan: (enum BtInquiryScanstatus)

Status	Meaning
0	= **BT_INQUIRY_CAN_NOT_POSSIBLE** Scanning Bluetooth devices is not possible.
1	= **BT_INQUIRY_SCAN_START** Scanning activated
2	= **BT_INQUIRY_SCAN_RESULT** Results of a scan; the address btaddr and device name devname of a found device are returned.
3	= **BT_INQUIRY_SCAN_BUSY** Scanning is still active
4	= **BT_INQUIRY_SCAN_TIMEOUT** Timeout when scanning; the end identifier was not received by the library
5	= **BT_INQUIRY_SCAN_END** Scanning stopped; the results are the Bluetooth devices reported in previous calls.

6.5 CancelScanBtDevice

DWORD CancelScanBtDevice (HANDLE fthdl)

Early cancellation of the scan for Bluetooth devices in the vicinity of the current ROBO TX Controller connected to the PC triggered by the StartScanBtDevice() function. No feedback is provided via the callback function.

Call parameters:

fthdl - valid handle of the local ROBO TX Controller

Return code:

errCode - FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

6.6 ConnectBtAddress

DWORD ConnectBtAddress (HANDLE fthdl,
DWORD chanIdx,
*BYTE *btaddr,*
*void (__stdcall *) cbFunc (BT_CB *data))*

Active establishment of a Bluetooth connection to a Bluetooth remote party uniquely identified by the Bluetooth target address. The result of the attempt to connect is reported asynchronously via the callback function. After a successful connection is made, this connection is managed in the library with the specified channel index (chanIdx, alias remote call number) for further accesses. To make multiple Bluetooth connections active simultaneously, this function can be called multiple times (each time with one channel index (chanIdx), alias remote call number).

Limitations:

The ConnectBtAddress function reports via the callback call an error (**BT_CON_CHANNEL_BUSY**) if on the same channel index an on-call duty status has already been registered via the function BtListenConOn(). This prevents multiple connections between two boards which could occur if a program initiates calls to another board while simultaneously also attempting to accept calls from the same board.

Call parameters:

fthdl	- valid handle of the local ROBO TX Controller
chanIdx	- channel index under which the connection is managed. Determined by the calling application (1 to 8).
*btaddr	- pointer to the associated Bluetooth address (6 bytes)
cbFunc	- callback function reporting the result of the connection

Return code:

errCode	- FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data	- pointer to data structure BT_CB

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Channel index
      UINT16      status;     //  Connection result
} BT_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS** , action successful, connection established
1	= **BT_CON_EXIST** , already connected
2	= **BT_CON_SETUP** , connection to this BT address is actively being carried out
3	= **BT_SWITCHED_OFF** , connection failed: Bluetooth is switched off locally as per configuration
4	= **BT_ALL_CHAN_BUSY** , connection failed: Bluetooth channel no longer available locally
5	= **BT_NOT_ROBOTX** , connection failed: incompatible BT device cannot be connected (not a ROBO-TX controller)
6	= **BT_CON_TIMEOUT** , failed: timeout, no device can be reached at this address (timeout)
12	**= BT_DISCON_INDICATION**, signals passive termination of a connection (e.g. triggered by remote party). The Bluetooth connection no longer exists.
14	**= BT_CHANNEL_BUSY**, active connection is not permitted when the listen function is activated on the same channel index.
15	**= BT_BTADDR_BUSY**, for this Bluetooth address, there is already an active connection via a different channel index. Dual connections are not permitted.
16	**= BT_NO_LISTEN_ACTIVE**, on the remote party no listen function was activated; connection is not possible.

6.7 BtListenConOn

DWORD BtListenConOn (HANDLE fthdl,
DWORD chanIdx,
*BYTE *btadr,*
*void (__stdcall *) cbFunc (BT_CB *data)*

Passive establishment of a Bluetooth connection by a Bluetooth remote party uniquely identified by the specified Bluetooth source address. In this case, the ROBO TX Controller signals it is ready to receive exactly one Bluetooth connection by the named remote party (activated on-call duty status). Via the callback function, the incoming connection is reported asynchronously, as soon as this status is reached, just as any possible errors are reported. After the connection is made, the connection is managed in the library with the specified channel index (chanIdx, alias remote call number) for further accesses. To receive multiple active Bluetooth connections at the same time, this function can be called multiple times (each time with one chanIdx, alias remote call number).

Limitations:

The ListenBtAddress function reports an error (**BT_CON_CHANNEL_BUSY**) via the callback call if on the same channel index there is already an active connection that was made via ConnectBtAddress(). This prevents multiple connections between two boards which could occur if a program initiates calls to another board while simultaneously also attempting to accept calls from the same board.

Call parameters:

fthdl - current handle of the local ROBO TX Controller
chanIdx - channel index under which the connection is managed. Determined by the calling application (1 to 8).
btadr - pointer to a valid Bluetooth address (6 bytes) or NULL
cbFunc - pointer of callback function that reports the result (message)

Return code:

errCode - FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data - pointer to data structure BT_CB

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Channel index
      UINT16      status;     //  Connection result
} BT_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS**, the listen function is activated.
3	= **BT_SWITCHED_OFF**, the listen function cannot be activated: Bluetooth is switched off locally as per configuration.
5	= **BT_NOT_ROBOTX**, listen function failed: BT address of incompatible BT device that cannot be connected (not a ROBO TX-C)
9	**= BT_LISTEN_ACTIVE**, the listen function has already been activated for the specified channel index.
11	**= BT_CON_INDICATION**, signals a connection on the passive side. A Bluetooth connection from the specified Bluetooth address is established.
12	**= BT_DISCON_INDICATION**, signals passive termination of a connection (e.g. triggered by remote party). The Bluetooth connection no longer exists.
14	**= BT_CHANNEL_BUSY**, passive connection is not permitted when a connection exists on the same channel index.
15	**= BT_BTADDR_BUSY**, for this Bluetooth address, there is already a listen registration (on-call duty) via a different channel index. Dual connections are not permitted.

6.8 BtListenConOff

DWORD BtListenConOff (HANDLE fthdl,
DWORD chanIdx,
*void (__stdcall *) cbFunc (BT_CB *data)*

This function deactivates an on-call duty status that was previously activated by the BtListenConOn() function on the specified channel index. This means that as of this moment, Bluetooth connections are no longer accepted. However, a Bluetooth connection that already exists on this channel index will not be dropped by this function call, but will instead remain intact. Ending the on-call duty in this case only has an effect on the period after the existing connection is ended (a connection can no longer be accepted until BtListenConOn() has been called again).

Call parameters:

fthdl - current handle of the local ROBO TX Controller
chanIdx - channel index (1 to 8), not available for loopback mode
cbFunc - pointer of callback function that reports the status (message)

Return code:

errCode - FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data - pointer to data structure (BT_CB)

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Channel index
      UINT16      status;     //  Connection result
} BT_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS**, the listen function has been deactivated.

6.9 DisconnectBt

DWORD DisconnectBt (HANDLE fthdl,
DWORD chanIdx,
*void (__stdcall *) cbFunc (BT_CB*data))*

Termination of an active Bluetooth connection referenced by the specified channel index. In this case it does not matter whether the connection was made actively or passively. The result of the attempt to disconnect is reported asynchronously via the callback function.

Call parameters:

fthdl	- valid handle of the local ROBO TX Controller
chanIdx	- channel index (1 to 8)
cbFunc	- callback function reporting the result of the disconnection

Return code:

errCode	- FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data	- pointer to data structure (BT_CB)

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Channel index
      UINT16      status;     //  Connection result
} BT_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS**, action successful, connection was successfully terminated
7	= **BT_CON_INVALID**, there is no active connection with the specified channel index
8	= **BT_CON_RELEASE**, termination of connection to this BT address is already activated and is being carried out

6.10 SendBtMessage

DWORD SendBtMessage (HANDLE fthdl,
DWORD chanIdx,
DWORD len,
LPSTR sendBuff,
*void (__stdcall *) cbFunc (BT_CB *data))*

Writes data to an active Bluetooth connection referenced by the channel index. This call is used to pass a pointer to a send data buffer and a length. The function reads the specified number of bytes from the send data buffer. After the function call, the send data buffer can be freed up again. The result of the send attempt is reported asynchronously via the callback function.

Call parameters:

fthdl	- current handle of the local ROBO TX Controller
chanIdx	- channel index (1 to 8)
*sendBuff	- pointer to send buffer with the send data (message)
len	- length of the send data in the send buffer (max. 255 characters)
cbFunc	- pointer of callback function that reports the result

Return code:

errCode	- FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data	- pointer to data structure (BT_CB)

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Channel index
      UINT16      status;     //  Result of the operation
} BT_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS**, data successfully recorded (confirmed)
7	= **BT_CON_INVALID**, there is no active connection with the specified channel index.

6.11 BtReadMsgOn

DWORD BtReadMsgOn (HANDLE fthdl,
DWORD chanIdx,
*void (__stdcall *) cbFunc (BT_RECV_CB *data)*

This function is used to display the ready receive status of data (messages) on an active Bluetooth connection referenced by the channel index. When receiving a message, the callback function is called which contains a pointer to the received data and the length of the data. For the ready receive status, this function must be called only once (per channel index). Accordingly, incoming data call the callback function multiple times.

Call parameters:

fthdl - current handle of the local ROBO TX Controller
chanIdx - channel index (1 to 8)
cbFunc - pointer of callback function that reports the result (message)

Return code:

errCode - FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data - pointer to data structure (BT_RECV_CB)

Data structure:

```
typedef struct {
      UINT16      chanIdx;    // Connection index
      UINT16      status;     // Read attempt result, enum CB_BtStatus
      UINT16      msglen;     // Length of the received message (max. 255)
      UCHAR8      *msg;       // Pointer to the received Bluetooth message
} BT_RECV_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS**, action successful, the message listener is activated, messages ready to be received from partner, length = 0
7	= **BT_CON_INVALID**, there is no active connection with the specified channel index, length = 0
10	= **BT_RECEIVE_ACTIVE**, the receive message function has already been activated for the specified channel index, length = 0
13	= **BT_MSG_INDICATION**, signals the receipt of a message from the remote party, length != 0

6.12 BtReadMsgOff

DWORD BtReadMsgOff (HANDLE fthdl,
DWORD chanIdx,
*void (__stdcall *) cbFunc (BT_CB *data)*

Deactivation of the activated ready receive status by the function *BtReadMsgOn()*. Calling this function prevents the receipt of data on the connection with the specified channel index. However, incoming data are dismissed in the interim by the ROBO TX Controller on any Bluetooth connection that may still exist.

Call parameters:

fthdl - valid handle of the local ROBO TX Controller
chanIdx - channel index (1 to 8)
cbFunc - pointer of callback function that reports the status

Return code:

errCode - FTLIB_ERR_SUCCESS (=0) or error code, see ftlib.h and ftMscLib.h

Callback function return codes:

*data - pointer to data structure (BT_CB)

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Channel index
      UINT16      status;     //  Result of the operation
} BT_CB;
```

Possible status codes:

Status	Meaning
0	= **BT_SUCCESS**, the ready receive status for incoming messages has been deactivated.

6.13 StatusBtConnection

*void StatusBtConnection (DWORD chanIdx, BT_STATUS *status)*

A library function that returns the (current) status of a Bluetooth connection for the specified channel index if the transfer area is not active. The local library information is returned via the BT_STATUS structure. The field strength is specified with 0. The remaining status codes are set by the library depending on the controller's reported status information. If the transfer area is activated, this function returns the current status information, including the field strength from the transfer area, for the specified channel index.

Call parameters:

chanIdx	- channel index (1 to 8)
BT_STATUS	- pointer to data structure

For the definition and meaning of the status codes, see section 6.15, "Status indicator in the transfer area".

6.14 List of status codes in the callback functions

List of possible status codes: (`enum CB_BtStatus`)

Status	Meaning
0	= **BT_SUCCESS** , action successful
1	= **BT_CON_EXIST** , already connected
2	= **BT_CON_SETUP** , connection to this BT address is actively being carried out
3	= **BT_SWITCHED_OFF** , connection failed: Bluetooth is switched off locally as per configuration
4	= **BT_ALL_CHAN_BUSY** , connection failed: Bluetooth channel no longer available locally
5	= **BT_NOT_ROBOTX** , connection failed: incompatible BT device cannot be connected (not a ROBO TX Controller)
6	= **BT_CON_TIMEOUT** , failed: timeout, no device can be reached at this address (timeout)
7	= **BT_CON_INVALID**, there is no active connection with the specified channel index.
8	= **BT_CON_RELEASE**, termination of connection to this BT address is already activated and is being carried out
9	**= BT_LISTEN_ACTIVE**, the listen function has already been activated for the specified channel index.
10	**= BT_RECEIVE_ACTIVE**, the receive function has already been activated.
11	**= BT_CON_INDICATION**, signals a connection on the passive side. A Bluetooth connection from the specified Bluetooth address is established.
12	**= BT_DISCON_INDICATION**, signals passive termination of a connection (e.g. triggered by remote party). The Bluetooth connection no longer exists.
13	**= BT_MSG_INDICATION**, signals the receipt of a message from the remote party
14	**= BT_CHANNEL_BUSY**, the specified channel index is already registered (on-call duty) or in use (active connection).
15	**= BT_BTADDR_BUSY**, for this Bluetooth address, there is already an on-call duty status or an active connection via a different channel index.
16	**= BT_NO_LISTEN_ACTIVE**, on the remote party no listen function was activated; connection is not possible.

6.15 Status indicator in the transfer area

For each channel index (1 to 8), there is a connection status structure in the transfer area in which information about the current connection status and the remote signal strength is stored. When starting and ending the transfer area, the values are initialized using 0.

Possible Bluetooth connection status codes:

```
enum BtConnState
{
    BT_STATE_IDLE = 0,     // BT channel is disconnected
    BT_STATE_CONN_ONGOING, // BT channel is being connected
    BT_STATE_CONNECTED,    // BT channel is connected
    BT_STATE_DISC_ONGOING  // BT channel is being disconnected
};
```

Data structure:

```
typedef struct {              // 8 bytes
    UINT16  ConState;         // Connection status (enum BtConnState)
    BOOL16  is_listen;        // if TRUE - BT channel is waiting for
                              // incoming connection (listening)
    BOOL16  is_receive;       // if TRUE - BT channel is ready to receive
                              // incoming messages
    UINT16  link_quality;     // 0...31 signal quality
                              // 0- the worst, 31- the best signal quality
} BT_STATUS;
```

The Bluetooth status indicator is integrated into the existing FTX1_STATE structure. The status information in the structure is updated every second when the transfer area is active.

```
#define BT_CNT_MAX      8

typedef struct
{
    // used by local application
    BOOL8           init;
    BOOL8           config;
    unsigned char   dummy[2];
    UINT32          trace;

    // public state info
    BOOL8           io_mode;
    UCHAR8          id;
    UCHAR8          info_id;
    UCHAR8          config_id;
    BOOL8           io_slave_alive[SLAVE_CNT_MAX];

    BT_STATUS       btstatus[BT_CNT_MAX];

    PGM_INFO        master_pgm;
    PGM_INFO        local_pgm;
} FTX1_STATE;
```

7 I²C API functions

A ROBO TX Controller can operate external sensors and actuators that have an I²C interface via the controller's I²C interface (EXT2 connection). They can be operated from a PC program via the I²C API functions described as follows.

The I²C interface and its pin assignment and internal wiring are described in the introductory chapter 7 (I²C interface) in the document "PC_Programming_RoboTXC.pdf", version 1.5.

7.1 Introduction

Each of the following function calls represents its own I²C bus access with the device address specification. This makes it possible to activate a variety of different I²C devices even within one program.

Important notes:

The I²C device address must be specified with only 7 bits in accordance with the I²C specification (value range: 0 to 127).

Moreover, I²C the device addresses 80 and 84 (0x50 and 0x54) are reserved for an internal EEPROM of the ROBO TX Controller. Access to these addresses is not permitted by the API. The I²C device addresses 81 through 83 and 85 through 87 (0x51 through 0x53 and 0x55 through 0x57) are also reserved memory areas for the same EEPROM, but are not used by the firmware. In this case, bus access conflicts could (but may not necessarily) occur with external I²C devices that use one of these addresses.

7.2 ftxI2cRead

```
DWORD  ftxI2cRead  (HANDLE fthdl,
                    BYTE DevAddr,
                    DWORD Offset,
                    BYTE Flags,
                    void (__stdcall *) cbFunc (I2C_CB *data))
```

One byte (8-bit) or two bytes (16-bit) is read on the I²C bus at the "DevAddr" device address and possibly within the device at the "Offset" subaddress. The addressing, data bus width, byte sequence (16-bit only), behavior in the case of bus errors and access speed are specified using the "Flags" parameter. Via the callback function, the result of the read access as well as the read datum is returned asynchronously as soon as this status is reached.

Call:	fthdl	- valid handle of the local ROBO TX Controller
	BYTE DevAddr	- 12C device address
	DWORD Offset	- if addressing is required within the device, then this internal address is passed here. The value of "Flags" specifies the length of the internal address in bits 0..1.

BYTE Flags - Access flags used:

Bit 0..1	Addressing	00: none ("Offset" invalid) 01: 8-bit addressing 10: 16-bit addressing, MSB first 11: 16-bit addressing, LSB first
Bit 2..3	Data width	00: - *not permitted* - 01: 8-bit data (1 byte) 10: 16-bit data (2 bytes), MSB first 11: 16-bit data (2 bytes), LSB first
Bit 4	KeepOpen	0: normal access 1: quick access without STOP/START
Bit 5..6	Error Mask = behavior in the case of bus error	00: abort 01: repeat up to 10 times 10: repeat until successful 11: - *not permitted* -
Bit 7	Clock rate	0: standard (100 kHz) 1: fast (400 kHz)

cbFunc - callback function that reports back the result of the operation asynchronously.

Return:

DWORD errCode - FTLIB_ERR_SUCCESS (no error) or error code for an error during the call (see 7.4).

Callback function return codes:

*data - pointer to data structure 12C_CB

Data structure:

```
typedef struct {              //  4 bytes
      UINT16      value;      //  Datum read (with 8-Bit, only LSByte is valid)
      UINT16      status;     //  Result of I2C bus operation (see 7.4)
} I2C_CB;
```

7.3 ftxI2cWrite

DWORD ftxI2cWrite (HANDLE fthdl,
BYTE DevAddr,
DWORD Offset,
WORD Data,
BYTE Protocol,
*void (__stdcall *) cbFunc (I2C_CB *data))*

One byte (8-bit) or two bytes (16-bit) is written on the I^2C bus at the "DevAddr" device address and possibly within the device at the "Offset" subaddress. The addressing, data width, byte sequence (16-bit only), behavior in the case of bus errors and access speed are specified using the "Flags" parameter. Via the callback function, the result of the write access as is returned asynchronously as soon as this status is reached.

Call:

fthdl	- valid handle of the local ROBO TX Controller
BYTE DevAddr	- 12C device address
DWORD Offset	- if addressing is required within the device, then this internal address is passed here. The value of "Protocol" specifies the length of the internal address
WORD Data	- datum (8-bit or 16-bit) to be written
BYTE Flags	- access flags used:

Bit 0..1	Addressing	00: none ("Offset" invalid) 01: 8-bit addressing 10: 16-bit addressing, MSB first 11: 16-bit addressing, LSB first
Bit 2..3	Data width	00: - *not permitted* - 01: 8-bit data (1 byte) 10: 16-bit data (2 bytes), MSB first 11: 16-bit data (2 bytes), LSB first
Bit 4	KeepOpen	0: normal access 1: quick access without STOP/START
Bit 5..6	Error Mask = behavior in the case of bus error	00: abort 01: repeat up to 10 times 10: repeat until successful 11: - *not permitted* -
Bit 7	Clock rate	0: standard (100 kHz) 1: fast (400 kHz)

cbFunc - callback function that reports back the result of the operation asynchronously.

Return:

DWORD errCode - FTLIB_ERR_SUCCESS (no error) or error code for an error during the call (see 7.4).

Callback function return codes:

*data - pointer to data structure 12C_CB

Data structure:

```
typedef struct {                  //  4 bytes
      UINT16      value;          //  Written datum repeated
      UINT16      status;         //  Result of I2C bus operation (see 7.4)
} I2C_CB;
```

7.4 Error codes of I²C API functions (errCode or status)

Return codes during call (errCode)

Value of errCode	Meaning
0x00000000	Function call successful FTLIB_ERR_SUCCESS
0xE0005000	Invalid device address FTLIB_I2C_INVALID_DEV_ADDR
0xE0005001	Invalid flag for address size FTLIB_I2C_INVALID_FLAGS_ADDRMODE
0xE0005002	Invalid flag for data width FTLIB_I2C_INVALID_FLAGS_DATAMODE
0xE0005003	Invalid flag for error mask (behavior in the event of bus error) FTLIB_I2C_INVALID_FLAGS_ERRMODE

Return codes during callback (status)

Status code	Meaning
0	I2C operation successful I2C_SUCCESS
1	I2C read error I2C_READ_ERROR
2	I2C write error I2C_WRITE_ERROR

8 Error codes

The ftErrCode.h include file contains error codes:

```
#define FTLIB_ERR_SUCCESS                       0x00000000L
#define FTLIB_ERR_NO_MEMORY                     0xE0000100L

#define FTLIB_ERR_FAILED                        0xE0001000L
#define FTLIB_ERR_TIMEOUT                       0xE000100CL
#define FTLIB_ERR_INVALID_PARAM                 0xE0001018L

#define FTLIB_ERR_SOME_DEVICES_ARE_OPEN         0xE0001101L
#define FTLIB_ERR_DEVICE_IS_OPEN                0xE0001102L
#define FTLIB_ERR_DEVICE_NOT_OPEN               0xE0001103L
#define FTLIB_ERR_NO_SUCH_DEVICE_INSTANCE       0xE0001104L

#define FTLIB_ERR_UNKNOWN_DEVICE_HANDLE         0xE0001283L
#define FTLIB_ERR_LIB_IS_INITIALIZED            0xE0001286L
#define FTLIB_ERR_LIB_IS_NOT_INITIALIZED        0xE0001287L
#define FTLIB_ERR_THREAD_NOT_STARTABLE          0xE00012A0L
#define FTLIB_ERR_THREAD_IS_RUNNING             0xE00012A5L
#define FTLIB_ERR_THREAD_NOT_RUNNING            0xE00012A6L
#define FTLIB_ERR_THREAD_SYNCHRONIZED           0xE00012AFL

#define FTLIB_ERR_TIMEOUT_TA                    0xE00012B0L
#define FTLIB_ERR_CREATE_EVENT                  0xE00012B1L
#define FTLIB_ERR_CREATE_MM_TIMER               0xE00012B2L

//  Upload files to ROBO TX Controller
#define FTLIB_ERR_UPLOAD_FILE_NOT_OPEN          0xE0001400L
#define FTLIB_ERR_UPLOAD_FILE_READ_ERR          0xE0001401L
#define FTLIB_ERR_UPLOAD_INVALID_FSIZE          0xE0001402L
#define FTLIB_ERR_UPLOAD_START                  0xE0001403L
#define FTLIB_ERR_UPLOAD_CANCELED               0xE0001404L
#define FTLIB_ERR_UPLOAD_FAILED                 0xE0001405L
#define FTLIB_ERR_UPLOAD_TIMEOUT                0xE0001406L
#define FTLIB_ERR_UPLOAD_ACK                    0xE0001407L
#define FTLIB_ERR_UPLOAD_NAK                    0xE0001408L
#define FTLIB_ERR_UPLOAD_DONE                   0xE0001409L
#define FTLIB_ERR_UPLOAD_FLASHWRITE             0xE000140AL
#define FTLIB_ERR_REM_CMD_FAILED                0xE000140BL
#define FTLIB_ERR_REM_CMD_NOT_SUPPORTED         0xE000140CL
#define FTLIB_ERR_FWUPD_GET_FILES               0xE000140DL
#define FTLIB_ERR_FWUPD_NO_FILES                0xE000140EL

//  Open connection to controller
#define FTLIB_ERR_ACCESS_DENIED                 0xE0001905L
#define FTLIB_ERR_OPEN_COM                      0xE0001906L
#define FTLIB_ERR_INIT_COM                      0xE0001908L
#define FTLIB_ERR_INIT_COM_TIMEOUT              0xE0001909L

#define FTLIB_ERR_WRONG_HOSTNAME_LEN            0xE0002000L
```

```
//  Firmware update
#define FTLIB_FWUPD_UPLOAD_START            0xE0003000L
#define FTLIB_FWUPD_UPLOAD_DONE             0xE0003001L
#define FTLIB_FWUPD_TIMEOUT                 0xE0003002L
#define FTLIB_FWUPD_FLUSH_DISK              0xE0003003L
#define FTLIB_FWUPD_CLEAN_DISK              0xE0003004L
#define FTLIB_FWUPD_ERR_FILE_READ           0xE0003005L
#define FTLIB_FWUPD_UPLOAD_FAILED           0xE0003006L
#define FTLIB_FWUPD_STARTING                0xE0003007L
#define FTLIB_FWUPD_FINISHED                0xE0003008L
#define FTLIB_FWUPD_REM_COMMAND             0xE0003009L
#define FTLIB_FWUPD_REM_TIMEOUT             0xE000300AL
#define FTLIB_FWUPD_REM_FAILED              0xE000300BL
#define FTLIB_FWUPD_IZ_STEPS                0xE000300CL
#define FTLIB_FWUPD_STEP                    0xE000300DL

//  Bluetooth
#define FTLIB_BT_INVALID_CONIDX             0xE0004000L
#define FTLIB_BT_CON_NOT_EXISTS             0xE0004001L
#define FTLIB_BT_CON_ACTIVE                 0xE0004002L
#define FTLIB_BT_CON_INACTIVE               0xE0004003L
#define FTLIB_BT_CON_WRONG_ADDR             0xE0004004L
#define FTLIB_BT_CON_WAIT_BUSY              0xE0004005L

//  I2C
#define FTLIB_I2C_INVALID_DEV_ADDR          0xE0005000L
#define FTLIB_I2C_INVALID_FLAGS_ADDRMODE    0xE0005001L
#define FTLIB_I2C_INVALID_FLAGS_DATAMODE    0xE0005002L
#define FTLIB_I2C_INVALID_FLAGS_ERRMODE     0xE0005003L

#define FTLIB_ERR_UNKNOWN                   0xEFFFFFFFL
```

9 Memory layout of transfer area

The transfer area of the "ftMscLib" library is divided into multiple data structures (see also the layout of the transfer area in Chapter 1, "General information"). For the comparison of the output and input structure, information is sent to the ROBO TX Controller and received, just like in the "old library via a communication thread. The comparison cycle is controlled via a multimedia timer and is 10 ms. The transfer area within the library has the same structure as the transfer area within the firmware on the controller.

The following describes only the structure variables in the individual structures that are important for communication with the ROBO TX Controller or for reading out particular information. All other fields are used by the firmware and cannot be changed by the library, nor read out.

Note: All definitions and data structures not listed here are defined in the following header files:

```
common.h
ftErrCode.h
ROBO_TX_FW.h
ftMscLib.h
```

.

9.1 FT_VERSION structure

Contains information on the versions of the ROBO TX Controller firmware and hardware.

```
// Versions of hardware and firmware components, 16 bytes
typedef struct
{
    FT_VER          hardware;
    FT_VER          firmware;
    FT_VER          ta;
    char            reserved[4];
} FT_VERSION;
```

Variable	Data type	Meaning
hardware	FT_VER	Hardware version (hardware.part.a = 'A' or 'B' or 'C')
firmware	FT_VER	Firmware version ("V %d.%02d, DLL %d", firmware.part.c, firmware.part.d, firmware.part.b)
ta	FT_VER	Version of transfer area ("V %d.%02d", ta.part.c, ta.part.d)

9.2 TA_INFO structure

Contains information on the ROBO TX Controller.

```
// Info structure, 64 bytes
```

```
typedef struct
{
    char            device_name[DEV_NAME_LEN_MAX + 1];
    char            bt_addr[BT_ADDR_STR_LEN + 1];
    char            reserved;
    UINT32          ta_array_start_addr;
    UINT32          pgm_area_start_addr;
    UINT32          pgm_area_size;
    FT_VERSION      version;
} TA_INFO;
```

Variable	Data type	Meaning
device_name	char[]	Stores the name of the ROBO TX Controller; can be changed using a library function
bt_addr	char[]	Stores the assigned Bluetooth address
ta_array_start_addr	UINT32	Starting address of the TransferAreaArray memory area
pgm_area_start_addr	UINT32	Program area starting address
pgm_area_size	UINT32	Program area length
version	FT_VERSION	Supplies the installed firmware and hardware version (see FT_VERSION structure)

9.3 BT_STATUS structure

Particular Bluetooth status information can be read.

```
// Bluetooth connection status structure, 8 bytes
typedef struct btstatus_s
{
    UINT16          conn_state;
    BOOL16          is_listen;
    BOOL16          is_receive;
    UINT16          link_quality;
} BT_STATUS;
```

Variable	Data type	Meaning
conn_state	UINT16	See enum BtConnState
is_listen	BOOL16	If TRUE - Bluetooth channel waits for incoming connection (listening)
is_receive	BOOL16	If TRUE - Bluetooth is ready to receive messages
link_quality	UINT16	0...31, 0 = weakest, 31 = best signal quality

9.4 TA_STATE structure

Particular status information can be read and stored. For a configuration comparison, e.g. the variable *config_id* is incremented and thus the communication thread automatically detects that the configuration in the transfer area is to be updated on the ROBO-TX controller. The variable is monitored constantly by the library.

```
// State structure, 100 bytes
typedef struct
{
    // Used by local program
    BOOL8           pgm_initialized;
    char            reserved_1[7];

    // Public state info
    BOOL8           dev_mode;
    UINT8           id;
    UINT8           info_id;
    UINT8           config_id;
    BOOL8           ext_dev_connect_state[N_EXT];
    BT_STATUS       btstatus[BT_CNT_MAX];
    char            reserved_2[8];
    PGM_INFO        local_pgm;
} TA_STATE;
```

Variable	Data type	Meaning
config_id	UINT8	Marks a configuration change (inputs, counter, etc.) in the transfer area; the configuration change is activated with the next comparison on the controller. Must be increased by 1 each time if something changed in the configuration.
ext_dev_connect_state	BOOL8[]	Identification for which slaves are active on the RS-485 bus
btstatus	BT_STATUS []	Status of Bluetooth connections (see BT_STATUS structure)

9.5 TA_CONFIG structure

The structure specifies the configuration of the motor outputs, universal inputs and the counter inputs. This structure is sent to the ROBO TX Controller each time the configuration is changed, activating the configuration of the inputs and outputs.

```
// Config structure, 88 bytes
typedef struct
{
    UINT8           pgm_state_req;
    char            reserved_1[3];
    BOOL8           motor[N_MOTOR];
    UNI_CONFIG      uni[N_UNI];
    CNT_CONFIG      cnt[N_CNT];
    char            reserved_2[32];
} TA_CONFIG;
```

Variable	Data type	Meaning
motor	BOOL8[]	Specifies if the outputs are defined as 4 motor outputs (=TRUE) or as 8 digital outputs (=FALSE, PWM outputs).
uni	UNI_CONFIG[]	Specifies the configuration of the 8 universal inputs: mode= 0, digital= TRUE - digital 10 V (trail sensor) mode= 0, digital= FALSE - analog 10 V (color sensor) mode= 1, digital= TRUE - digital 5 kOhm (push-button switch) mode= 1, digital= FALSE - analog 5 kOhm (NTC, etc.) mode= 3, digital= unimportant - ultrasonic distance sensor
cnt	CNT_CONFIG[]	Specifies the configuration of the counter input: mode= 0 - normal (change from 0 -> 1) mode= 1 - inverse (change from 1 -> 0)

9.6 TA_INPUT structure

For each cyclical comparison, the FTX1_INPUT structure is transferred from the ROBO TX Controller to the library. At the time of the comparison, this contains all current input values and internal flags for controlling the motor.

```
// Input structure, 68 bytes
typedef struct
{
    INT16           uni[N_UNI];
    INT16           cnt_in[N_CNT];
    INT16           counter[N_CNT];
    INT16           display_button_left;
    INT16           display_button_right;
    BOOL16          cnt_resetted[N_CNT];
    BOOL16          motor_pos_reached[N_MOTOR];
    char            reserved[16];
} TA_INPUT;
```

Variable	Data type	Meaning
uni	INT16[]	Contains for each universal input the value currently determined by the firmware.
cnt_in	INT16[]	Current configuration of the counter input (normal, inverse)
counter	INT16[]	Current status of counter
display_button_left	INT16[]	Time measurement while push-button switch remains depressed. The counter status provides the time in 10 ms intervals, where the push-button switch remains depressed; 0= push-button switch is not depressed.
display_button_right	INT16[]	See above, like the left push-button switch
cnt_resetted	BOOL16[]	Is set to 1 if the last counter reset is successful.
motor_pos_reached	BOOL16[]	Flags for controlling the motor; for each motor, it provides notification if the target was reached according to the specifications (Flag !=0).

9.7 TA_OUTPUT structure

The content of the TA_OUTPUT structure is transferred from the library to the ROBO TX Controller with each cyclical comparison and updates the pre-defined values at the controller outputs.

```
// Output structure, 44 bytes
typedef struct
{
    UINT16          cnt_reset_cmd_id[N_CNT];
    UINT8           master[N_MOTOR];
    INT16           duty[N_PWM_CHAN];
    UINT16          distance[N_MOTOR];
    UINT16          motor_ex_cmd_id[N_MOTOR];
} TA_OUTPUT;
```

Variable	Data type	Meaning
cnt_reset_cmd_id	UINT16[]	Requests a counter reset. Must be increased by 1 each time if a reset is required.
master	UINT8[]	In the case of active motor control, the ID is determined for the slave motor from the master motor. The master ID in this case is the motor index (0 to 3) + 1. Example: Synchronized motor control with motor 1 (master, index= 0) and motor 3 (slave, index= 2). Contents: master[2] = 1
duty	INT16[]	Current duty values for the motor outputs; if the motor is actively switched on (*motor* variable from TA_CONFIG structure is set to TRUE), each in pairs, the difference of both values provides the real duty value corresponding to the direction of rotation (CW, CCW) duty[0] => M1- , duty[1] => M1+ duty[2] => M2- , duty[3] => M2+ etc. In PWM mode (digital output), the outputs can be operated independently of each other. duty[0] => O1 duty[1] => O2 duty[2] => O3 etc.
distance	UINT16[]	Counter value The value is the position at which the motor is to stop during active motor control. Motor control is activated only once the counter value is != 0
motor_ex_cmd_id	UINT16[]	Must always be increased by 1 if the settings for the active motor control (duty and/or distance) changed.

9.8 TA_DISPLAY structure

The TA_DISPLAY.DISPLAY_MSG structure is used to display generated text messages from an application on the ROBO TX Controller display. The output text remains visible until either the text field is initialized with 0 or the text is deleted by pressing one of the two push-button switches on the controller.

```
// Display structure, 108 bytes
typedef struct
{
    DISPLAY_MSG     display_msg;
    DISPLAY_FRAME   display_frame;
} TA_DISPLAY;


// Display frame, 8 bytes
// Used to refresh boards display with a bitmap image frame
typedef struct
{
    unsigned char   * frame;  // contents of a frame as a 128x64 pixels bitmap
    UINT16          id;       // should be increased by 1 each time a new
                              // display frame is to be shown

    BOOL16          is_pgm_master_of_display;

                              // ++ if program wants to have control over display,
                              // i.e. image frame is displayed over firmware
                              // menus;
                              // -- if program wants to return control over
                              // display to the firmware menus
} DISPLAY_FRAME;


// Display message, 128 bytes.
// Used to show pop-up message box on the boards display
typedef struct
{
    UINT8           id;
    char            text[DISPL_MSG_LEN_MAX + 1];
} DISPLAY_MSG;
```

Variable	Data type	Meaning
id	UINT8	Message ID. Must be increased by 1 each time if a new message window is to be displayed.
text	char[]	Character field for receiving text (ASCII) that is shown on the ROBO TX Controller display. Maximum text length is DISPL_MSG_LEN_MAX=98

9.9 TA_STATUS structure

Particular transfer area status information can be read.

```
// Status of transfer area (valid only for ftMscLib), 4 bytes
typedef struct
{
    UINT8           status;
    UINT8           iostatus;
    UINT16          ComErr;
} TA_STATUS;
```

Variable	Data type	Meaning
status	UINT8	Status of transfer area: 0 - transfer area is not active 1 - transfer area is active 2 - transfer area is being synchronized
iostatus	UINT8	Status of I/O communication: 0 - remote I/O request was sent 1 - configuration data sent
ComErr	INT16	System error message in the case of a COM port error

9.10 TA_CHANGE structure

Structure serves to provide notification of changes to the universal and counter inputs. If any change has been detected, the variable *ChangeStatus* is set to TRUE; if it is FALSE, there are no current changes. This allows an application to display the existing values in an optimal way.

```
// Change structure (valid only for ftMscLib), 8 bytes
typedef struct
{
    char            reserved_1[2];
    UINT8           ChangeStatus;
    UINT8           ChangeUni;
    UINT8           ChangeCntIn;
    UINT8           ChangeCounter;
    char            reserved_2[2];
} TA_CHANGE;
```

Variable	Data type	Meaning
ChangeStatus	UINT8	Global display of a change for universal inputs, counter inputs or time variables. TRUE - a change is present FALSE - the relevant values have not been changed in the transfer area
ChangeUni	UINT8	Each set bit within the UINT8 variables indicates that a change in the existing value has occurred for the representative universal input. Bit 0 - universal input I1 Bit 1 - universal input I2, etc.
ChangeCntIn	UINT8	Flags for displaying a configuration change to the counter inputs Bit 0 - counter input 1 Bit 1 - counter input 2, etc.
ChangeCounter	UINT8	Flags for displaying a change in the counter status Bit 0 - counter 0 Bit 1 - counter 2, etc.

9.11 TA_TIMER structure

Timer variables are managed in the new transfer area as they were in the old transfer area. These have the same purpose as in the previous version and are used for certain timeout variables. The timer variables are updated in an event-controlled manner via a multimedia timer.

```
// 16-bit timers, 12 bytes
typedef struct
{
    UINT16          Timer1ms;
    UINT16          Timer10ms;
    UINT16          Timer100ms;
    UINT16          Timer1s;
    UINT16          Timer10s;
    UINT16          Timer1min;
} TA_TIMER;
```

10 Document change history

Version	Date	Author	Other comments
1.4.24	9/21/2009	Peter Classen	First complete documented version
1.5.08	3/17/2011	Peter Duchemin	Added Bluetooth message API functions
1.5.11	4/24/2012	Peter Duchemin	Added I^2C API functions