Programmierung des ROBO TX Controllers

Teil 2: Windows Library "ftMscLib"

Library V1.5.11 vom 19.02.2012
(Dokument-Version = Library-Version)

Referenzen:

Bezeichnung	Version	Datum
ROBO TX Controller Firmware	1.30	19.03.2012
ROBOPro	3.1.3	26.03.2012
Package „PC_Programmierung_RoboTXC"	1.5	24.04.2012

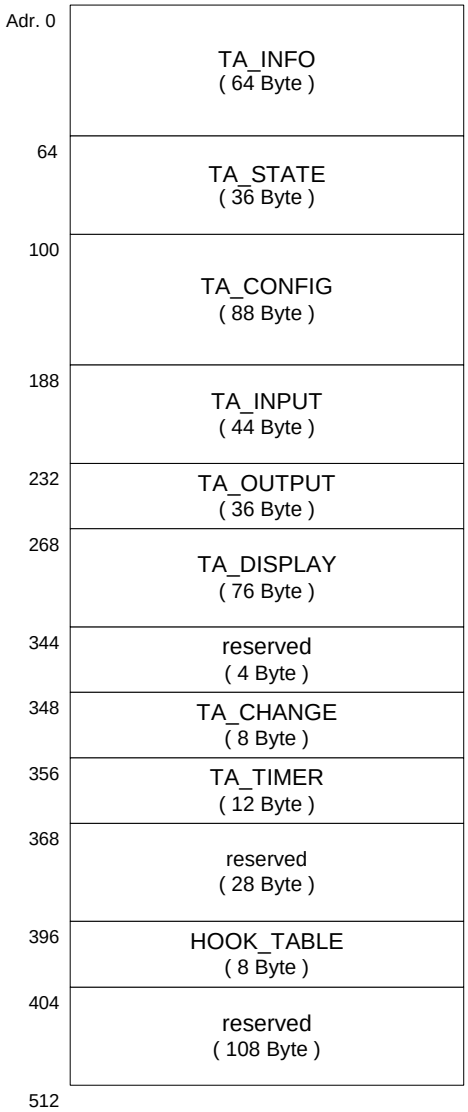
MSC Vertriebs GmbH
Design Center Aachen
Pascalstr. 21
52076 Aachen

Inhalt

1 Allgemeines - TransferArea

Für die Ansteuerung des ROBO TX Controllers wird wie bei der bisherigen Library *FtLib* von der Firma Knobloch und dem Vorgänger *ROBO Interface*, eine sogenannte TransferArea benutzt. In dieser TransferArea werden die Werte für die Aus- und Eingänge gesetzt oder ausgelesen. Ein Kommunikation-Thread gleicht diese Werte dann mit dem ROBO TX Controller ab, bzw. aktualisiert in der TransferArea die empfangenen Werte vom ROBO TX Controller. Diese TransferArea, sowie die Konfiguration des Kommunikations-Thread verwaltet die Library ftMscLib. Als API stellt die Library ein Satz von Funktionen zur Verfügung, die jeweils die Output- und Konfigurationswerte in der TransferArea setzen oder die vorhandenen aktuellen Input-Werte des ROBO TX Controllers ausliest.

Aufbau der neuen TransferArea in der ftMscLib und Firmware:



Stand V1.04.19 (2009/09/01)

Den Aufbau des Speicherlayouts wird im Kapitel 9 genauer beschrieben.

Um größere Modelle zu steuern, ist es evtl. erforderlich mehrere ROBO TX Controller zu koppeln. Dies wird über die integrierten „Extension-Ports“ durch eine Master/Slave-Struktur realisiert. In einer Kette von gekoppelten Controllern gibt es immer einen Master und bis zu 8 Slaves, die vom Master ferngesteuert werden.

Jeder ROBO TX Controller besitzt 2 Extension Ports, die einen seriellen Erweiterungsbus darstellen. Mit einem 6-poligen Flachbandkabel wird eine Verbindung von einem Controller zum jeweils benachbarten Controller hergestellt. Dadurch wird eine Datenverbindung vom Master zu jedem der Slaves realisiert.

Bei dem Erweiterungsbus handelt es sich um eine RS485-Schnittstelle mit Abgriffen auf jedem Controller. Zwischen den beiden Anschlüssen auf dem Controller ist der Erweiterungsbus durchverbunden, womit eine einfache Verkabelung benachbarter Controller möglich ist.

Die Kommunikation auf dem Erweiterungsbus ist ebenfalls Master/Slave orientiert. Der Master pollt dabei alle angeschlossenen Slaves (Extension-Controller) zyklisch ab und tauscht die I/O-Informationen aus.

Um dies zu realisieren, belegt jeder Controller in seinem Memory einen Satz von 9 TransferAreas, Master und 8 Slaves. Die eigene TransferArea wird dabei immer als Master gekennzeichnet und liegt am Anfang des belegten Speicherbereichs. Die Slaves werden vom Master im Online-Modus so gesteuert, wie ein einzelner Controller vom PC aus im Online-Modus gesteuert wird. Der Controller welcher als Master deklariert ist, ist der hierbei der einzige, der mit dem PC kommunizieren kann. Anfragen im Online-Modus vom PC an einem Slaves, werden beim Master über die entsprechende TransferArea und schließlich über den Erweiterungsbus an den entsprechenden Slave-Controller weitergeleitet.

Dies hat zur Folge, dass für viele Funktionsaufrufe eine eindeutige Controller-ID mit angegeben werden muss, damit die entsprechende Anfrage den richtigen Controller über die dazugehörige TransferArea auf dem Master-Controller erreicht.

Zur eindeutigen Adressierung eines Controllers sind dabei folgende Definitionen (Controller-ID) festgelegt worden:
(aus ROBO_TX_FW.h)

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

2 Allgemeine Funktionen der Library "ftMscLib"

2.1 ftxGetLibVersion

DWORD ftxGetLibVersion (void)

Versionsnummer der aktuellen Library als DWORD-Wert (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)

Die Versionsnummer der Library wird als String zurückgegeben.
Format "MM.RR.SS yyyy/mm/dd"
MM = Main-Version, RR = Release-Version, SS = Sub-Version, yyyy/mm/dd = Datum

Aufruf:	LPSTR strBuff	- Pointer Stringbuffer zur Aufnahme des Versionsstrings
	DWORD maxLen	- reservierte Länge des Stringbuffers
Return:	DWORD len	- Länge des Versionsstrings oder 0

2.3 ftxInitLib

DWORD ftxInitLib (void)

Initialisierungsfunktion der Library. Um die Funktionalität der Library zu nutzen, muss diese Funktion zuerst aufgerufen werden. Dabei wird das Logging vorbereitet, die internen Variablen und die FishX1-TransferArea werden initialisiert.

Return:	DWORD errCode	- Error-Code (ftErrCode.h)
	FTLIB_ERR_SUCCESS	- kein Fehler
	FTLIB_ERR_LIB_IS_INITIALIZED	- Library ist bereits initialisiert
	FTLIB_ERR_NO_MEMORY	- fehlendes Memory

2.4 ftxCloseLib

DWORD ftxCloseLib (void)

Gegenstück zur Funktion InitFtLib(), gibt den reservierten Speicher wieder frei. Schließt die Library.

Return:	DWORD errCode	- Error-Code (ftErrCode.h)
	FTLIB_ERR_SUCCESS	- kein Fehler

2.5 ftxIsLibInit

DWORD ftxIsLibInit (void)

Liefert Informationen, ob die Library ftMscLib initialisiert ist.

Return:	DWORD errCode	- Error-Code (ftErrCode.h)
	FTLIB_ERR_SUCCESS	- kein Fehler
	FTLIB_ERR_LIB_IS_INITIALIZED	- Library ist initialisiert
	FTLIB_ERR_LIB_IS_NOT_INITIALIZED	- Library ist nicht initialisiert

2.6 ftxOpenComDeviceNr

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

Die Funktion öffnet zur Kommunikation mit dem ROBO TX Controller die mit einer Nummer angegebene COM-Schnittstelle und liefert einen eindeutigen Handle zurück. Mögliche Werte für eine COM-Schnittstelle: 1 bis 255. Die zur Verfügung stehenden Anschlussnummern sind aus der Auflistung im Geräte-Manager zu entnehmen. Die Variable *errcode* dient zur Aufnahme eines möglichen Fehlercodes.

Aufruf:	DWORD port	- Portnummer der COM-Schnittstelle, z.B. 12 für COM12
	DWORD baudr	- Baudrate, z. Zt. 38400 fest
	DWORD *errcode	- Zeiger auf eine Error-Variable
Return:	HANDLE fthdl	- Handle zur Kommunikation mit dem ROBO TX Controller, bei einem aufgetretenen Fehler (=NULL) enthält die Variable *errcode* dann einen möglichen Error-Code.

Möglicher Error-Code (ftErrCode.h)

FTLIB_ERR_SUCCESS	- kein Fehler (Handle != NULL)
FTLIB_ERR_DEVICE_IS_OPEN	- COM ist bereits geöffnet
FTLIB_ERR_SOME_DEVICES_ARE_OPEN	- eine andere COM ist geöffnet
FTLIB_ERR_OPEN_COM	- Fehler beim Öffnen einer COM

2.7 ftxOpenComDevice

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

Funktion äquivalent zur Funktion aus 2.6, jedoch kann die entsprechende COM-Schnittstelle als String übergeben werden, z.B. "COM4" oder "COM32".

Aufruf:	char comStr	- COM-Schnittstelle mit Portnummer als String, z.B. "COM4"
	DWORD baudr	- Baudrate, z. Zt. 38400 fest
	DWORD *errcode	- Zeiger auf eine Error-Variable

Return: siehe 2.6.

2.8 ftxCloseDevice

DWORD ftxCloseDevice (HANDLE fthdl)

Die Funktion schließt eine geöffnete Verbindung zum angegebenen ROBO TX Controller. Alle aktiven Threads zur Kommunikation mit dem ROBO TX Controller werden gestoppt.

Aufruf:	HANDLE fthdl	- gültiges Handle einer COM-Schnittstelle
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

2.9 ftxCloseAllDevices

DWORD ftxCloseAllDevices (void)

Funktion schließt alle geöffneten Verbindungen zu allen möglichen ROBO TX Controllern. Zurzeit wird nur eine Verbindung zu einem ROBO TX Controller unterstützt. Beim Aufruf der Funktion wird somit die einzige vorhandene Verbindung geschlossen.

Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

2.10 ftxIsHandleValid

DWORD ftxIsHandleValid (HANDLE fthdl)

Überprüft, ob das angegebene Handle einer COM-Schnittstelle (noch) gültig ist.

Aufruf: HANDLE fthdl - gültiges Handle einer COM-Schnittstelle

Return: DWORD errCode - Error-Code (ftErrCode.h)

FTLIB_ERR_SUCCESS	- Handle ist gültig, kein Fehler
FTLIB_ERR_DEVICE_NOT_OPEN	- eine COM ist nicht geöffnet
FTLIB_ERR_UNKNOWN_DEVICE_HANDLE	- das angegeben Handle ist ungültig oder nicht bekannt

2.11 GetComStatus

DWORD GetComStatus (HANDLE fthdl)

Gleiche Funktionalität wie die Funktion "ftxIsHandleValid()", siehe 2.10

2.12 GetAvailableComPorts

DWORD GetAvailableComPorts (int selectMode)

Funktion ermittelt auf Basis der Informationen aus der Windows-Registry alle zur Verfügung stehenden COM-Ports und liefert die Anzahl der Ports zurück. Intern werden diese Informationen in einer statischen Liste zur weiteren Verwendung verwaltet. Zur Ermittlung und Anzeige der zur Verfügung stehenden COM-Ports muss diese Funktion zuerst aufgerufen werden.

Aufruf: int selectMode - Angabe, ob alle COM-Ports gelesen werden oder nur die Ports, die für eine Verbindung zu einem ROBO-TX Controller vorgesehen sind (USB oder Bluetooth)

ALL_PORTS	0	alle Ports
USB_ONLY	1	nur USB-Schnittstellen
BT_ONLY	2	nur Bluetooth-Verbindungen

Return: DWORD iComPorts - Anzahl der ermittelten COM-Ports in der internen Liste

2.13 EnumComPorts

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

Liefert einen Eintrag aus der von der Library erstellten statischen Liste der zur Verfügung stehenden COM-Ports. Ein vorheriger Aufruf der Funktion "GetAvailableComPorts()" ist notwendig. Der Eintrag wird als String mit Angabe des COM-Ports und einer Beschreibung der Verwendung, wie sie aus der Windows-Registry zu entnehmen ist, zurückgegeben, z.B.: "COM32 (fischertechnik USB ROBO TX Controller)". Der übergebene Stringbuffer muß entsprechend dimensioniert sein, damit dieser den String mit der Beschreibung aufnehmen kann.

Aufruf:	DWORD idx	- Listenindex innerhalb der Liste der verfügbaren COM-Ports 0 < idx < iComPorts (GetAvailableComPorts())
	LPSTR comstr	- Stringbuffer zur Aufnahme der COM-Beschreibung
	DWORD maxLen	- max. Länge des Stringbuffers
Return:	DWORD index	- Index des Listeneintrags welcher gelesen wurde (= idx), im Fehlerfall wird der Wert FTLIB_ERR_INVALID_PARAM zurückgegeben

2.14 ftxGetLibErrorString

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

Funktion liefert entweder eine repräsentative Fehlermeldung anhand des übergebenen Error-Codes oder die Fehlerkonstante selber als String zurück. Damit kann eine Fehleranalyse gemacht werden, auch wenn der Fehlercode nicht bekannt ist.

Aufruf:	DWORD errCode	- Fehlercode
	DWORD typ	- Typ des zurückgegebenen String 0 = Fehlerkonstante als String 1 = englischer Fehlertext
	LPSTR strBuff	- Buffer zur Aufnahme der Fehlermeldung (nullterm. String)
	DWORD maxLen	- Länge des Aufnahmebuffers
Return:	DWORD len	- Länge der kopierten Fehlermeldung oder 0

2.15 ftxGetManufacturerStrg

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

Die Funktion liefert einen nullterminierten String mit den Angaben zum Hersteller zurück, z. Zt. noch statisch "MSC Vertriebs GmbH".

Hinweis: Diese Information wird z. Zt. noch nicht aus dem ROBO TX Controller ausgelesen.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Herstellerangaben
	DWORD maxlen	- verfügbare Länge von *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.16 ftxGetShortNameStrg

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

Die Funktion liefert einen nullterminierten String mit der offiziellen Bezeichnung "ROBO TX Controller".

Hinweis: Diese Information wird z. Zt. noch nicht aus dem ROBO TX Controller ausgelesen.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Zeichenfolge
	DWORD maxlen	- verfügbare Länge von *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.17 ftxGetLongNameStrg

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

Ruft die Funktion "GetFtShortNameStrg()" auf, Parameterbeschreibung siehe 2.16

2.18 ftxGetFirmwareStrg

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

Diese Funktion liefert einen nullterminierten String mit der auf dem ROBO TX Controller installierten Firmware zurück. Format: 'V xx.yy' .

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Zeichenfolge
	DWORD maxlen	- verfügbare Länge von *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.19 ftxGetSerialNrStrg

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

Diese Funktion liefert einen nullterminierten String mit der Seriennummer des angeschlossenen ROBO TX Controllers zurück.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Zeichenfolge
	DWORD maxlen	- verfügbare Länge von *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.20 GetRoboTxDevName

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

Funktion liefert einen nullterminierten String mit dem Namen des angegebenen ROBO TX Controllers. Die Funktion kopiert das Ergebnis in den Stringbuffer *strBuff*, der entsprechend dimensioniert sein muss. Die Länge des Namens beträgt max. 16 Zeichen. Standardmäßig setzt sich der Name des ROBO TX Controllers aus der Zeichenkette "ROBO TX-" und einer 3-stelligen Nummer "xxx" zusammen. Die vergebene Nummer ist die Quersumme der vergebenen Bluetooth-Adresse als Dezimalzahl.

Beispiel: Aus der vorhandene Bluetooth-Adresse des Controllers "00:13:7B:B2:16" ergibt sich die Quersumme "1A8" (hex), entspricht "424" (dezimal). Der Name des vorliegenden ROBO TX Controllers lautet somit: "ROBO TX-424", welche standardmäßig auf dem Display des Controllers erscheint.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer zur Aufnahme des Controllernamens
	DWORD len	- max. Länge des übergebenen Stringbuffers *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.21 SetRoboTxDevName

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

Mit dieser Funktion kann der standardmäßig vergebene Name eines ROBO TX Controllers geändert werden. Die Länge des Hostnamen beträgt max. 16 Zeichen.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer der den neuen Namen des Controllers beinhaltet
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

2.22 GetRoboTxBtAddr

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

Diese Funktion liefert einen nullterminierten String mit der vergebenen Bluetooth-Adresse des angeschlossenen ROBO TX Controllers zurück. Die maximale Länge des String zur Aufnahme der Bluetooth-Adresse beträgt 17 Zeichen.

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

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Bluetooth-Adresse
	DWORD len	- max. Länge des übergebenen Stringbuffers *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.23 GetRoboTxFwStr

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

Diese Funktion liefert einen nullterminierten String mit der Version der Firmware, die augenblicklich auf dem ROBO TX Controller installiert ist. Format: 'V xx.yy' .

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Firmware-Version
	DWORD len	- max. Länge des übergebenen Stringbuffers *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.24 GetRoboTxFwVal

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

Diese Funktion liefert die Firmwareversion als DWORD zurück. Damit kann die Firmwareversion verglichen werden. Der Wert setzt sich aus der Version der Library ROBOLIB.dll (xx) und der Version der Firmware (yy.zz) zusammen:

	00	xx	yy	zz
Byte	4	3	2	1

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	DWORD *version	- Variable zur Aufnahme eines Wertes zur Firmware
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

2.25 GetRoboTxHwStr

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

Diese Funktion liefert einen nullterminierten String mit der vorhandenen Hardware-Release Version zurück.

Beispiel: 'C'

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Hardware Release
	DWORD len	- max. Länge des übergebenen Stringbuffers *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.26 GetRoboTxSerialStr

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

Diese Funktion liefert einen nullterminierten String mit der Seriennummer des angeschlossenen ROBO TX Controllers zurück.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Hardware Release
	DWORD len	- max. Länge des übergebenen Stringbuffers *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

2.27 GetRoboTxDllStr

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

Diese Funktion liefert einen nullterminierten String mit der Version der ROBO-TX Library „ROBOLIB.DLL“, die auf dem Controller abgelegt ist. Format: 'xx' .

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPSTR strBuff	- Stringbuffer zur Aufnahme der Hardware Release
	DWORD len	- max. Länge des übergebenen Stringbuffers *strBuff*
Return:	DWORD len	- Länge der kopierten Zeichenfolge, bei Fehler = 0

3 Funktionen für den Online-Betrieb

3.1 ftxStartTransferArea

DWORD ftxStartTransferArea (HANDLE fthdl)

Funktion aktiviert die TransferArea in der Library für den Online-Betrieb. Der Kommunikations-Thread wird gestartet und führt die I/O-Befehle im „Online-Modus“ durch. Dabei liest dieser die aktuellen Werte aus der Output-Struktur der TransferArea (Konfiguration- und Outputwerte) und sendet diese zum ROBO TX Controller. Als Antwort auf einen I/O-Request sendet der Controller die aktuellen Werte und der Kommunikations-Thread aktualisiert diese dann in der Input-Struktur der TransferArea.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler, Thread ist aktiviert) oder Error-Code

3.2 ftxStopTransferArea

DWORD ftxStopTransferArea (HANDLE fthdl)

Funktion deaktiviert die Kommunikation der TransferArea mit dem ROBO TX Controller. Der Kommunikations-Thread wird gestoppt. Es findet keine Kommunikation mehr mit dem ROBO TX Controller statt.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler, Thread wurde gestoppt) oder Error-Code

3.3 ftxIsTransferActiv

DWORD ftxIsTransferActiv (HANDLE fthdl)

Funktion prüft, ob die TransferArea aktiv ist, d.h. ob eine zyklische Kommunikation mit dem ROBO TX Controller stattfindet.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
Return:	DWORD errCode	- Error-Code (ftErrCode.h)
FTLIB_ERR_THREAD_NOT_RUNNING		- Transfer-Thread nicht aktiv
FTLIB_ERR_THREAD_SYNCHRONIZED		- Thread synchronisiert sich mit dem ROBO TX-C
FTLIB_ERR_THREAD_IS_RUNNING		- Thread ist aktiv im Online-Modus

3.4 GetTransferAreasArrayAddr

*volatile TA_ARRAY *GetTransferAreasArrayAddr (HANDLE fthdl)*

Die Funktion liefert einen Zeiger auf den Speicherbereich aller TransferAreas zurück (TransferAreas sind als Array von Strukturen organisiert).

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
Return:	TA_ARRAY * adr	- Anfangsadresse den Speicherbereich aller TransferAreas

Alle TransferAreas (Master + bis zu 8 Slaves) sind in diesem Speicherbereich abgelegt. Der Aufbau der TransferArea ist im Kapitel 1 beschrieben. Der Aufbau der Strukturen innerhalb der TransferArea ist der Header-Datei ROBO_TX_FW.h zu entnehmen.

3.5 GetTransferAreaStatusAddr

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

Die Funktion gibt einen Zeiger auf die Struktur TA_STATUS innerhalb der TransferArea zurück. Aus dieser kann der aktuelle Status der TransferArea gelesen werden.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
Return:	TA_STATUS * adr	- Zeiger auf die Struktur TA_STATUS. Die Struktur ist in der Header-Datei ROBO_TX_FW.h definiert.

Mögliche Statusmeldungen:

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

3.6 StartCounterReset

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

Funktion startet die Prozedur zum Rücksetzen des Zählereingang auf dem ROBO TX Controller auf 0 zurück. Dies ist ein asynchroner Vorgang und ist daher nicht direkt nach Rückkehr des Funktionsaufrufs erledigt. Eine Bestätigung kann nach Bedarf per Callback erfolgen. Hierzu ist eine Callback-Funktion mit SetCBCounterResetted (s.u.) zu setzen.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int shmId	- Controller-ID (Master- oder Extension-Controller)
	int cntId	- Counterindex (0 – 3) der Counter 1 - 4
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.7 SetCBCounterResetted

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

Funktion installiert in der Library die angegebene Callback-Funktion, die den Status "Zählereingang wurde zurückgesetzt" meldet.

Paramter der Callback-Funktion:

Aufruf:	DWORD devId	- Controller-ID (Master- oder Extension-Controller)
	DWORD cntId	- Counterindex (0 – 3) der Counter 1 - 4

3.8 SetOutMotorValues

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

Funktion setzt in der TransferArea für einen Motor die Duty-Werte für die beiden Motorausgänge M+ und M-.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int motorId	- Index des anzusteuernden Motors (0 - 3)
	int duty_p	- Duty-Wert für den Motorausgang M+
	int duty_m	- Duty-Wert für den Motorausgang M-
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.9 SetOutPwmValues

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

Funktion setzt in der TransferArea für einen PWM-Ausgang den angegebenen Duty-Wert.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int outId	- Index des PWM-Ausgang (0-7)
	int duty	- Duty-Wert für den PWM-Ausgang
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.10 SetFtUniConfig

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

Funktion konfiguriert einen Universaleingang (Kombi-Eingang) zur Messung von analogen und digitalen Spannungs- und Widerstandswerten und zur analogen Abstandsmessung.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int ioId	- Index des Universaleingangs (0 - 7)
	int mode	- Art der Messung 0= Spannung (mV), 1= Widerstand (5 kΩ) 3= Ultraschallsensor (Abstandsmessung)
	BOOL digital	- Kennung ob der Wert digital zurückgegeben wird (FALSE= analog, TRUE= digital), bei der Abstandsmessung nur analog
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.11 SetFtCntConfig

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

Funktion konfiguriert einen Zählereingang (Counter), wie der Zustand des Zählers zu interpretieren ist.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int cntId	- Counter-Index (0 - 3)
	int mode	- 0= Mode NORMAL, 1= Mode INVERTED
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.12 SetFtMotorConfig

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

Funktion aktiviert oder deaktiviert die Motorausgänge. Analog dazu werden damit die PWM-Ausgänge entsprechend deaktiviert oder aktiviert.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int motorId	- Motorindex (0 - 3)
	BOOL status	- TRUE= Motor an (Motorausgänge aktiviert)
		FALSE= Motor aus (PWM-Ausgang aktiviert)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.13 StartMotorExCmd

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

Funktion aktiviert den intelligenten Motor-Modus zur Motorsynchronisation. Der Motor fährt die angestrebte Position anhand der mitgeteilten Zählerinformation an. Das Erreichen der Endposition wird der Applikation mittels einer vorher installierten Callback-Funktion mitgeteilt (s. a. SetCBMotorExReached()).

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int mIdx	- Motorindex (0 - 3) vom Master (-motor)
	int duty	- Duty-Wert für Master/Slave-Motor
	int mDirection	- Richtung für Master-Motor (0= CW, 1= CCW)
	int sIdx	- Motorindex (0 – 3) vom Slave (-motor)
	int sDirection	- Richtung für Slave-Motor (0= CW, 1= CCW)
	int pulseCnt	- Anzahl der Zählimpulse zum Anfahren einer Position, relativ zur Ausgangsposition
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.14 StopMotorExCmd

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

Funktion stoppt unverzüglich den vorher aktivierten intelligenten Motor-Modus zur Motorsynchronisation für den angegebenen Motor durch Zurücksetzen auf 0 der Duty-Werte sowie des Distance-Wertes in der Output-Struktur.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int mtrIdx	- Motorindex (0 - 3)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.15 StopAllMotorExCmd

DWORD StopAllMotorExCmd (HANDLE fthdl,
int devId)

Funktion stoppt unverzüglich den vorher aktivierten intelligenten Motor-Modus zur Motorsynchronisation für alle Motoren des ROBO TX Controllers durch Zurücksetzen auf 0 der Duty-Werte sowie der Distance-Werte in der Output-Struktur.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.16 SetCBMotorExReached

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

Funktion installiert in der Library die angegebene Callback-Funktion, die den Status "Motor Reached State" bei einer aktiven Motorsynchronisation (intelligenter Motor-Modus) meldet.

Paramter der Callback-Funktion:

Aufruf:	DWORD devId	- Controller-ID (Master- oder Extension-Controller)
	DWORD mtrIdx	- Motorindex (0 – 3)

3.17 GetInIOValue

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

Funktion liest aus der TransferArea den aktuellen Wert eines Universaleingangs und stellt diesen einer Applikation zur Verfügung. Die Werte der Universaleingänge kommen als Antwort auf einen IO-Request vom ROBO TX Controller und werden in der TransferArea aktualisiert.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int ioId	- Index Universaleingang (0 - 7)
	INT16 *value	- Zeiger auf Variable, die den Wert aufnimmt
	BOOL32 *overrun	- Zeiger auf eine Variable für Overrun-Meldung FALSE: kein Überlauf (Overrun) TRUE : Überlauf
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.18 GetInCounterValue

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

Funktion liest aus der TransferArea den aktuellen Wert eines Zählereingangs (Counter) und stellt diesen einer Applikation zur Verfügung.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	int cntId	- Index Zähler (Counter) (0 - 3)
	INT16 *count	- Zeiger auf Variable, aktueller Zählerstand
	INT16 *state	- Zeiger auf Variable, eingestellter Mode des Zählers TRUE : Mode "INVERTED" FALSE: Mode "NORMAL"
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.19 GetInDisplayButtonValue

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

Funktion liest den aktuellen Status der beiden Taster auf dem Display. Angezeigt wird die Zeit, wie lange ein Taster gedrückt bleibt.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	INT16 *left	- Zeiger auf INT16-Variable, Taster [Links]
	INT16 *right	- Zeiger auf INT16-Variable, Taster [Rechs]
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.20 FtRemoteCmd

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

Die Funktion sendet einen Konsolenbefehl zum ROBO TX Controller. Die Antwort vom ROBO TX Controller wird der Callback-Funktion als Stringbuffer übergeben und kann entsprechend ausgewertet werden. Zusätzlich wird noch die Länge der empfangenen Daten bereitgestellt.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	char *ftCmd	- Zeiger auf einen nullterminierten String mit einem (Remote-) Konsolenbefehl
	cbFunc	- Funktionspointer einer Callback-Funktion zur Anzeige oder Darstellung der empfangenen Remote-Daten vom ROBO TX Controller
	LPSTR strBuff	- Pointer Stringbuffer mit Remote-Daten
	DWORD len	- Länge der Remote-Daten
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.21 RTxCleanDisk

DWORD RTxCleanDisk (HANDLE fthdl,
DWORD device)

Funktion bereinigt das angegebene Laufwerk auf dem ROBO TX Controller. Das Bereinigen des Laufwerks wird nur für die Laufwerke RAMDISK und FLASH unterstützt. Ein Löschen aller Einträge auf der SYSTEM Partition wird nicht unterstützt. Auf dem FLASH Laufwerk verbleibt die Datei "sys_par.ini" auch nach dem Bereinigen des Laufwerks.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	DWORD device	- Ziellaufwerk (ramdisk=0, flash=1)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.22 GetRoboTxMemLayout

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

Funktion liest aus dem ROBO TX Controller die benötigten Adressen für ein ROBOPro-Programm. Alle Adressen sind im Adressraum der CPU des ROBO TX Controllers (nicht im PC-Adressraum).

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	PULONG pTAarray	- Anfangsadresse des TransferAreaArray-Speicherbereichs
	PULONG pAppStart	- ApplicationAreaStart Adresse
	PULONG pAppSize	- ApplicationArea Länge
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

3.23 SetCBRoboExtState

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

Funktion installiert in der Library eine Callback-Funktion, die Statusmeldungen von externen ROBO TX Controllern meldet, die sich im Multi-Controller-Modus befinden (Master/Slave – Betrieb). Über die Callback-Funktion wird gemeldet, ob ein Slave-Controller online oder offline, bzw. mit dem Master verbunden oder nicht verbunden ist.

Aufruf:	cbFunc	- Funktionspointer einer Callback-Funktion
	DWORD devId	- Controller-ID eines Slave-Controllers
	DWORD state	- Status, ob SLAVE_OFFLINE=0 oder SLAVE_ONLINE=1

3.24 SetRoboTxMessage

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

Funktion generiert einen darstellbaren Text auf das Display des ROBO TX Controllers. Der Text kann auf dem Display nur dargestellt werden, wenn die TransferArea aktiviert ist. Die maximale Textlänge beträgt 98 Zeichen (ASCII). Bei der Übergabe eines leeren Texts (= 0), wird der aktuell angezeigte Text auf dem Display gelöscht und es erscheint wieder die Standardausgabe des ROBO TX Controllers. Alternativ kann über die beiden Taster auf dem Controller die angezeigten und gepufferten Meldungen gelöscht werden.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	int devId	- Controller-ID (Master- oder Extension-Controller)
	LPCSTR msg	- Text, der auf dem Display dargestellt wird, max. 98 Zeichen
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

4 Funktionen für den Upload von Dateien

4.1 FtFileUpload

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

Funktion führt einen Upload der angegebenen Datei zum ROBO TX Controller durch. Die Datei wird dabei mit dem vollständigen Pfad angegeben. Eine anzugebende Callback-Funktion informiert über den Zustand, bzw. Status der Übertragung.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	char *fname	- Zeiger auf einen nullterminierten String mit dem vollständigen Pfadnamen der Datei oder des Programms
	DWORD device	- Ziellaufwerk (ramdisk=0, flash=1, system=2)
	cbFunc	- Funktionspointer einer Callback-Funktion

Return: DWORD errCode - FTLIB_ERR_SUCCESS (kein Fehler) Thread, der das Upload ausführt, ist aktiviert oder Error-Code

Mögliche Statuswerte für die Callback-Funktion:

`FTLIB_ERR_UPLOAD_START`	- Upload wurde gestartet
`FTLIB_ERR_UPLOAD_TIMEOUT`	- Upload endet mit einem Timeout
`FTLIB_ERR_UPLOAD_CANCELED`	- Upload wurde von der Gegenseite beendet
`FTLIB_ERR_UPLOAD_FAILED`	- Upload wurde wegen eines Fehlers beendet
`FTLIB_ERR_UPLOAD_FILE_READ_ERR`	- Fehler beim Lesen der Datei aufgetreten, Upload wurde beendet
`FTLIB_ERR_UPLOAD_NAK`	- Empfang einer NAK-Kennung (0x15) Paket wird wiederholt (bis zu 5 Versuche)
`FTLIB_ERR_UPLOAD_ACK`	- Empfang einer ACK-Kennung (0x06), nächstes Paket wird gesendet
`FTLIB_ERR_UPLOAD_DONE`	- Upload wurde erfolgreich durchgeführt
`FTLIB_ERR_UPLOAD_FLASHWRITE`	- Flash wird beschrieben

4.2 FtRamFileUpload

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

Funktion führt ein Upload eines Programms oder Datei zum ROBO TX Controller durch. Angegeben wird dabei ein Zeiger auf einen Memory-Bereich, der die zu übertragenden

Daten enthält, und die Länge der zu übertragenen Bytes. Auf dem ROBO TX Controller wird die Datei oder das Programm unter dem beim Aufruf mit anzugebenden Namen auf dem entsprechenden Laufwerk gespeichert. Die Callback-Funktion informiert über den Zustand, bzw. Status während der Übertragung.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	DWORD device	- Ziellaufwerk (ramdisk=0, flash=1, system=2)
	CONST PVOID pProg	- Pointer auf das Programm zum Upload
	DWORD size	- zu übertragene Programmlänge
	LPSTR progname	- nullterminierter String mit dem Programmnamen
	cbFunc	- Funktionspointer einer Callback-Funktion
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) Thread, der das Upload ausführt, ist aktiviert oder Error-Code

Mögliche Statuswerte der Callback-Funktion siehe 4.1.

4.3 RoboTxFwUpdate

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

Die Funktion erzeugt einen Thread in der Library, der ein komplettes Firmware-Update auf dem ROBO TX Controller durchführt. Die zur Firmware-Update benötigten Dateien liegen im angegebenen Verzeichnis. Über die installierte Callback-Funktion wird der aufrufenden Applikation Informationen bezüglich des Verlaufs mitgeteilt.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controllers
	LPCSTR path	- Verzeichnis, welches die Dateien zum Firmware-Update beinhaltet
	cbFunc	- Funktionspointer einer Callback-Funktion
	DWORD status	- Status Firmware-Update Definitionen s. u. 6 (Fehlercodes)
	LPSTR infoStr	- String mit Informationen über die ausgeführte Aktion während des Firmware-Updates
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code

5 Funktionen zur Programmsteuerung

5.1 FtProgramRun

DWORD FtProgramRun (HANDLE fthdl,
int devId)

Funktion startet ein geladenes Programm. Das Programm kann z.B. über die Funktion FtFileUpload() in den ROBO TX Controller geladen werden. Anschließend kann das gespeicherte Programm über diese Funktion gestartet werden.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controller
	int devId	- Controller-ID (Master- oder Extension-Controller)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler, Programm wurde gestartet) oder Error-Code

5.2 FtProgramStop

DWORD FtProgramStop (HANDLE fthdl,
int devId)

Funktion stoppt und beendet das laufende Programm auf dem ROBO TX Controller. Das Programm kann danach wieder über die Funktion FtProgramRun() gestartet werden.

Aufruf:	HANDLE fthdl	- aktueller Handle des ROBO TX Controller
	int devId	- Controller-ID (Master- oder Extension-Controller)
Return:	DWORD errCode	- FTLIB_ERR_SUCCESS (kein Fehler, Programm wurde gestoppt) oder Error-Code

6 Funktionen der Bluetooth-Message-API

Ein ROBO TX-Controller kann programmgesteuert Bluetooth-Verbindungen zu anderen ROBO TX-Controllern aufbauen und über diese kurze Textnachrichten (z.B. 10 Bytes) schicken. Programme können dann z.B. über Funk verschiedene ROBO TX Controller synchronisieren.

Es können auch zusätzliche Informationen (z.B. die aktuelle Funksignalstärke) angezeigt werden und es kann insbesondere auf Funkereignisse (z.B. ein Verbindungsabbruch, weil der Roboter außer Reichweite fährt) reagiert werden.

Es sind zur Nutzung der Bluetooth-Messaging-Funktionen sind mindestens zwei ROBO TX Controller erforderlich. Zum grundsätzlichen Aufbau einer Testumgebung sei auch das einführende Kapitel 6 zum Thema „Bluetooth Messaging API“ im Dokument „PC_Programmierung_RoboTXC.pdf“ Version 1.5 empfohlen.

6.1 Einführung

Es werden in der Library maximal 8 Verbindungen verwaltet. Jede Verbindung wird über einen sogenannten Kanalindex referenziert, der durch die Applikation in einem bestimmten Wertebereich vorgegeben wird. Beim Aufbau einer Bluetooth-Verbindung wird, nach erfolgreichem Aufbau der Verbindung, die Bluetooth-Adresse mit dem anzugebenden Kanalindex verknüpft. Die weiteren Zugriffe auf eine Verbindung werden dann nur noch über den Kanalindex getätigt. Die Applikation (hier: das C-Programm) verwaltet allein die Zuordnung Kanalindex <−> Bluetooth-Adresse.

Jeder Funktionsaufruf mit einem Zugriff auf eine Bluetooth-Verbindung hat als Parameter eine Callback-Funktion. Die Library verwaltet diese Callback-Funktionen für jeden Kanalindex und ruft diese im Bedarfsfall auf. Verwaltet wird für die jeweilige Funktion immer die zuletzt zu einer Verbindung angegebene Callback-Funktion.

6.2 Rufbereitschaft und Empfangsbereitschaft

Der ROBO TX-Controller nimmt nicht automatisch Bluetooth-Verbindungen an, sondern erst, wenn ein Programm gestartet ist und dieses zudem seine Rufbereitschaft mit der Funktion BtListenConOn() signalisiert hat. Wenn kein Programm gestartet ist oder das Programm so geschrieben ist, dass die Rufbereitschaft nicht von Beginn an besteht (sondern z.B. konditional erst zu einem späteren Zeitpunkt), dann wird der Verbindungsversuch von einer Gegenstelle mit Statuscode 16 (BT_NO_LISTEN_ACTIVE) abgelehnt.

Ebenso verhält es sich mit der Empfangsbereitschaft von Nachrichten. Erst nachdem bei einer bestehenden Bluetooth-Verbindung die Empfangsbereitschaft durch Aufruf der Funktion BtReadMsgOn() aktiviert wurde, werden auch empfangene Nachrichten gemeldet. Anderfalls werden empfangene Nachrichten bereits in der Firmware verworfen.

6.3 Loopback-Funktion

Zum lokalen Ausprobieren der Bluetooth-Message-API auf einem einzelnen ROBO TX Controller gibt es die Loopback-Funktion. Hierbei wird keine Funkverbindung zwischen zwei Controllern aufgebaut, sondern dieser Betrieb lokal nur simuliert.

Bei einem Scan-Vorgang mit StartScanBtDevice() wird zunächst immer die eigene Bluetooth-Adresse des lokalen Controllers gemeldet. Baut man eine Verbindung zu der eigenen Bluetooth-Adresse mit ConnectBtAddress() auf, so kommt sofort eine (simulierte) Bluetooth-Verbindung zustande. Ebenso beim Einschalten der Rufbereitschaft auf der eigenen Bluetooth-Adresse mit BtListenConOn(). Die der eigenen Bluetooth-Adresse zugeordnete Kanalnummer kann eine beliebige aus dem Wertevorrat 1-8 sein.

Werden auf einer Loopback-Verbindung mit SendBtMessage() Nachrichten versendet, so werden diese lokal in der Firmware des Controllers als Echo zurückgeschickt und sogleich auf derselben Verbindung (Kanal) wieder empfangen. Voraussetzung ist aber auch im Loopback-Betrieb, dass vorher die Empfangsbereitschaft durch Aufruf der Funktion BtReadMsgOn() aktiviert wurde.

6.4 StartScanBtDevice

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

Suchen nach Bluetooth-Geräten in der Umgebung des aktuell mit dem PC verbundenen ROBO TX Controllers (nicht verwechseln mit dem Suchen in der Umgebung des PC’s). Mit diesem Befehl wird der Suchvorgang nach verfügbaren Bluetooth-Geräten gestartet. Die aufgerufene Callback-Funktion liefert dann sukzessive Ergebnisse dieser Suche (ermittelte Bluetooth-Geräten) und kann mehrmals aufgerufen werden. Der gesamte Suchvorgang bis zur Meldung des Status BT_INQUIRY_SCAN_END dauert ca. 30 Sekunden. Durch Aufruf der Funktion CancelScanBtDevice() kann der Vorgang auch vorzeitig abgebrochen werden.

Aufruf-Parameter:

fthdl	- gültiger Handle des lokalen ROBO-TX Controllers
cbFunc	- Callback-Funktion, die das Ergebnis eines Scan-Vorgangs der Applikation meldet (Liste der gefundenen Bluetooth-Geräten)

Return-Wert:

errCode	- FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

result	- Pointer auf die Datenstruktur BT_SCAN_STATUS z.B. mit einem ermittelten Bluetooth-Gerät (status=BT_INQUIRY_SCAN_RESULT)

Datenstruktur:

```
typedef struct {
      UINT16      status;                     // Status Suchvorgang
      UCHAR8      btaddr[6];                  // Bluetooth-Adresse
      BYTE        dummy_1[2];
      Char        devname[HOSTNAME_LEN + 1];  // Gerätename
      BYTE        dummy_2[3];
} BT_SCAN_STATUS;
```

Mögliche Statuswerte nach dem Suchvorgang: (enum BtInquiryScanstatus)

Status	Bedeutung
0	= **BT_INQUIRY_CAN_NOT_POSSIBLE** Ein Scanning von Bluetooth-Geräten ist nicht möglich
1	= **BT_INQUIRY_SCAN_START** Scan-Vorgang wurde aktiviert
2	= **BT_INQUIRY_SCAN_RESULT** Ergebnis eines Scan-Vorgangs, die Adresse btaddr und der Gerätename devname eines gefundenen Gerätes werden zurückgeliefert.
3	= **BT_INQUIRY_SCAN_BUSY** Der Scan-Vorgang ist noch aktiv
4	= **BT_INQUIRY_SCAN_TIMEOUT** Timeout beim Scan-Vorgang, die Endekennung ist von der Library nicht empfangen worden
5	= **BT_INQUIRY_SCAN_END** Der Scan-Vorgang wurde beendet, Ergebnis sind die in vorherigen Aufrufen mitgeteilten Bluetooth-Geräte.

6.5 CancelScanBtDevice

DWORD CancelScanBtDevice (HANDLE fthdl)

Vorzeitiger Abbruch des durch die Funktion StartScanBtDevice() angestoßenen Suchvorgangs nach Bluetooth-Geräten in der Umgebung des aktuell mit dem PC verbundenen ROBO TX Controllers. Es erfolgt keine Rückmeldung per Callback-Funktion.

Aufruf-Parameter:

fthdl - gültiger Handle des lokalen ROBO-TX Controllers

Return-Wert:

errCode - FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

6.6 ConnectBtAddress

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

Aktiver Aufbau einer Bluetooth-Verbindung zu einer durch die Bluetooth-Ziel-Adresse eindeutig beschriebenen Bluetooth-Gegenstelle. Über die Callback-Funktion wird das Resultat des Verbindungsversuches asynchron gemeldet. Nach erfolgreichem Verbindungsaufbau wird diese Verbindung in der Library mit dem angegebenen Kanalindex (chanIdx, alias Funkrufnummer) für weitere Zugriffe verwaltet. Um mehrere Bluetooth-Verbindungen gleichzeitig aktiv aufzubauen, kann diese Funktion mehrfach aufgerufen werden (jeweils mit eigenem Kanalindex (chanIdx), alias Funkrufnummer).

Einschränkungen:

Die Funktion ConnectBtAddress meldet per Callback-Aufruf einen Fehler zurück (**BT_CON_CHANNEL_BUSY**), wenn auf demselben Kanalindex bereits eine Rufbereitschaft über die Funktion BtListenConOn() registriert wurde. Dies soll verhindern, dass Doppelverbindungen zwischen zwei Boards entstehen können, dadurch dass ein Programm von derselben Gegenstelle sowohl Rufe annimmt, als auch dorthin aufbaut.

Aufruf-Parameter:

fthdl	- gültiger Handle des lokalen ROBO-TX Controllers
chanIdx	- Kanalindex, unter der die Verbindung verwaltet wird. Wird von der aufrufenden Applikation festgelegt (1 – 8).
*btaddr	- Pointer auf die dazugehörige Bluetooth-Adresse (6 Byte)
cbFunc	- Callback-Funktion, die das Ergebnis des Verbindungsaufbaus meldet

Return-Wert:

errCode - FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsaufbaus
} BT_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS** , Aktion erfolgreich, Verbindung aufgebaut
1	= **BT_CON_EXIST** , Bereits verbunden
2	= **BT_CON_SETUP** , Verbindungsaufbau zu dieser BT-Adresse wird bereits aktiv durchgeführt
3	= **BT_SWITCHED_OFF** , Verbindung fehlgeschlagen: Bluetooth ist lokal per Konfiguration abgeschaltet
4	= **BT_ALL_CHAN_BUSY** , Verbindung fehlgeschlagen: kein Bluetooth-Kanal lokal mehr frei
5	= **BT_NOT_ROBOTX** , Verbindung fehlgeschlagen: nicht verbindbares fremdes BT-Gerät (kein ROBO-TX Controller)
6	= **BT_CON_TIMEOUT** , fehlgeschlagen: Timeout, kein Gerät unter dieser Adresse erreichbar (Timeout)
12	**= BT_DISCON_INDICATION**, Signalisierung eines passiven Verbindungsabbaus (z.B. von der Gegenseite ausgelöst). Die Bluetooth-Verbindung besteht nicht mehr.
14	**= BT_CHANNEL_BUSY**, aktiver Verbindungsaufbau wird nicht erlaubt bei aktivierter Listen-Funktion auf demselben Kanalindex.
15	**= BT_BTADDR_BUSY**, zu dieser Bluetooth Adresse existiert bereits eine aktive Verbindung über einen anderen Kanalindex. Doppelverbindungen sind nicht erlaubt.
16	**= BT_NO_LISTEN_ACTIVE**, auf der Gegenstelle ist keine Listen-Funktion aktiviert worden, kein Verbindungsaufbau möglich.

6.7 BtListenConOn

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

Passiver Aufbau einer Bluetooth-Verbindung von einer durch die angegebene Bluetooth-Quell-Adresse eindeutig beschriebenen Bluetooth-Gegenstelle. Hiermit signalisiert der ROBO TX Controller seine Bereitschaft genau eine Bluetooth-Verbindung von der genannten Gegenstelle anzunehmen (aktivierte Rufbereitschaft). Über die Callback-Funktion wird die eingehende Verbindung asynchron gemeldet, sowie dieser Zustand eintritt, ebenso wie eventuell auftretende Fehler. Nach erfolgtem Verbindungsaufbau wird die Verbindung in der Library mit dem angegebenen Kanalindex (chanIdx, alias Funkrufnummer) für weitere Zugriffe verwaltet. Um mehrere Bluetooth-Verbindungen gleichzeitig annehmen zu können, kann diese Funktion mehrfach aufgerufen werden (jeweils mit eigenem chanIdx, alias Funkrufnummer).

Einschränkungen:

Die Funktion ListenBtAddress meldet per Callback-Aufruf einen Fehler zurück (**BT_CON_CHANNEL_BUSY**), wenn auf demselben Kanalindex bereits eine aktive Verbindung besteht, die durch ConnectBtAddress() zustande kam. Dies soll verhindern, dass Doppelverbindungen zwischen zwei Boards entstehen können, dadurch dass ein Programm von derselben Gegenstelle sowohl Rufe annimmt, als auch dorthin aufbaut.

Aufruf-Parameter:

fthdl	- gültiger Handle des lokalen ROBO TX Controllers
chanIdx	- Kanalindex, unter der die Verbindung verwaltet wird. Wird von der aufrufenden Applikation festgelegt (1 – 8).
btadr	- Pointer auf eine gültige Bluetooth-Adresse (6 Byte) oder NULL
cbFunc	- Pointer der Callback-Funktion, die das Ergebnis (Nachricht) mitteilt

Return-Wert:

errCode	- FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data	- Pointer auf Datenstruktur BT_CB

Datenstruktur:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsaufbaus
} BT_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS**, die Listen-Funktion ist aktiviert.
3	= **BT_SWITCHED_OFF**, die Listen-Funktion kann nicht aktiviert werden: Bluetooth ist lokal per Konfiguration abgeschaltet.
5	= **BT_NOT_ROBOTX**, Listen-Funktion fehlgeschlagen: BT-Adresse von nicht verbindbarem fremdem BT-Gerät (kein ROBO TX-C)
9	**= BT_LISTEN_ACTIVE**, die Listen-Funktion ist für den angegebenen Kanalindex bereits aktiviert worden.
11	**= BT_CON_INDICATION**, Signalisierung eines Verbindungsaufbaus auf der passiven Seite. Eine Bluetooth-Verbindung von der vorgegebenen Bluetooth-Adresse ist eingegangen.
12	**= BT_DISCON_INDICATION**, Signalisierung eines passiven Verbindungsabbaus (z.B. von der Gegenseite ausgelöst). Die Bluetooth-Verbindung besteht nicht mehr.
14	**= BT_CHANNEL_BUSY**, passiver Verbindungsaufbau wird nicht erlaubt bei Bestehen einer Verbindung auf demselben Kanalindex.
15	**= BT_BTADDR_BUSY**, zu dieser Bluetooth Adresse existiert bereits eine Listen-Registrierung (Rufbereitschaft) über einen anderen Kanalindex. Doppelverbindungen sind nicht erlaubt.

6.8 BtListenConOff

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

Diese Funktion deaktiviert eine vorher durch die Funktion BtListenConOn() aktivierte Rufbereitschaft auf dem angegebenen Kanalindex. Das bedeutet, dass ab sofort keine Bluetooth-Verbindungen mehr angenommen werden. Allerdings wird eine bereits bestehende Bluetooth-Verbindung auf diesem Kanalindex durch diesen Funktionsaufruf nicht unterbrochen, sondern bleibt bestehen. Die Beendigung der Rufbereitschaft wirkt sich in diesem Fall nur auf den Zeitraum nach Beendigung der bestehenden Verbindung aus (es wird keine Verbindung mehr erneut angenommen bis nicht wieder BtListenConOn() aufgerufen wurde).

Aufruf-Parameter:

fthdl - gültiger Handle des lokalen ROBO TX Controllers
chanIdx - Kanalindex (1 – 8), für Loopback-Betrieb nicht verfügbar
cbFunc - Pointer der Callback-Funktion, die den Status (Nachricht) mitteilt

Return-Wert:

errCode - FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur (BT_CB)

Datenstruktur:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsaufbaus
} BT_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS**, die Listen-Funktion ist deaktiviert worden.

6.9 DisconnectBt

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

Abbau einer durch den angegebenen Kanalindex referenzierten aktiven Bluetooth-Verbindung. Hierbei ist es unerheblich, ob die Verbindung durch einen aktiven oder durch einen passiven Verbindungsaufbau zustande gekommen ist. Über die Callback-Funktion wird das Resultat des Verbindungsabbauversuches asynchron gemeldet.

Aufruf-Parameter:

fthdl - gültiger Handle des lokalen ROBO-TX Controllers
chanIdx - Kanalindex (1 bis 8)
cbFunc - Callback-Funktion, die das Ergebnis des Verbindungsabbaus meldet

Return-Wert:

errCode - FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur (BT_CB)

Datenstruktur :

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis des Verbindungsaufbaus
} BT_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS**, Aktion erfolgreich, Verbindung ist erfolgreich abgebaut worden
7	= **BT_CON_INVALID**, es existiert keine aktive Verbindung mit dem angegebenen Kanalindex
8	= **BT_CON_RELEASE**, Verbindungsabbau zu dieser BT-Adresse ist bereits aktiviert und wird durchgeführt

6.10 SendBtMessage

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

Schreiben von Daten auf eine durch den Kanalindex referenzierte aktive Bluetooth-Verbindung. Mit dem Aufruf wird ein Zeiger auf einen Schreibdatenpuffer und eine Länge übergeben. Die Funktion wird aus dem Puffer die angegebene Anzahl von Bytes auslesen. Nach dem Funktionsaufruf kann der Schreibdatenpuffer wieder freigegeben werden. Über die Callback-Funktion wird das Resultat des Sendeversuches asynchron gemeldet.

Aufruf-Parameter:

fthdl	- gültiger Handle des lokalen ROBO TX Controllers
chanIdx	- Kanalindex (1 bis 8)
*sendBuff	- Pointer auf Sendebuffer mit den Sendedaten (Nachricht)
len	- Länge der Sendedaten im Sendebuffer (max. 255 Zeichen)
cbFunc	- Pointer der Callback-Funktion, die das Ergebnis mitteilt

Return-Wert:

errCode	- FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data	- Pointer auf Datenstruktur (BT_CB)

Datenstruktur :

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis der Operation
} BT_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS**, Schreiben der Daten erfolgreich (bestätigt)
7	= **BT_CON_INVALID**, es existiert keine aktive Verbindung mit dem angegebenen Kanalindex.

6.11 BtReadMsgOn

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

Mit dieser Funktion wird die Empfangsbereitschaft von Daten (Nachrichten) auf eine durch den Kanalindex referenzierte aktive Bluetooth-Verbindung angezeigt. Beim Empfang einer Nachricht wird die Callback-Funktion aufgerufen, die einen Zeiger auf die empfangen Daten und die Länge der Daten enthält. Zur Empfangsbereitschaft muss diese Funktion nur 1 mal (pro Kanalindex) aufgerufen werden. Eingehende Daten rufen entsprechend mehrfach die Callback-Funktion auf.

Aufruf-Parameter:

fthdl - gültiger Handle des lokalen ROBO TX Controllers
chanIdx - Kanalindex (1 bis 8)
cbFunc - Pointer der Callback-Funktion, die das Ergebnis (Nachricht) mitteilt

Return-Wert:

errCode - FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur (BT_RECV_CB)

Datenstruktur:

```
typedef struct {
      UINT16      chanIdx;    // Verbindungsindex
      UINT16      status;     // Ergebnis des Leseversuchs, enum CB_BtStatus
      UINT16      msglen;     // Länge der empfangenen Message (max. 255)
      UCHAR8      *msg;       // Zeiger auf die empf. Bluetooth Nachricht
} BT_RECV_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS**, Aktion erfolgreich, der Message-Listener ist aktiviert, bereit Messages vom Partner zu empfangen, Länge = 0
7	= **BT_CON_INVALID**, es existiert keine aktive Verbindung mit dem angegebenen Kanalindex, Länge = 0
10	= **BT_RECEIVE_ACTIVE**, die Receive-Message Funktion ist für den angegebenen Kanalindex bereits aktiviert worden, Länge = 0
13	= **BT_MSG_INDICATION**, Signalisiert den Empfang einer Message von der Gegenstelle, Länge != 0

6.12 BtReadMsgOff

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

Deaktivierung der durch die Funktion *BtReadMsgOn()* aktivierten Empfangsbereitschaft. Durch Aufruf dieser Funktion wird der Datenempfang auf der Verbindung mit dem angegebenen Kanalindex unterbunden. Zwischenzeitlich dennoch eingehende Daten auf der ggf. noch bestehenden Bluetooth-Verbindung, werden vom ROBO TX Controller verworfen.

Aufruf-Parameter:

fthdl - gültiger Handle des lokalen ROBO-TX Controllers
chanIdx - Kanalindex (1 bis 8)
cbFunc - Pointer der Callback-Funktion, die den Status mitteilt

Return-Wert:

errCode - FTLIB_ERR_SUCCESS (=0) oder Fehlercode siehe ftlib.h und ftMscLib.h

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur (BT_CB)

Datenstruktur:

```
typedef struct {              //  4 bytes
      UINT16      chanIdx;    //  Kanalindex
      UINT16      status;     //  Ergebnis der Operation
} BT_CB;
```

Mögliche Statuswerte :

Status	Bedeutung
0	= **BT_SUCCESS**, die Empfangsbereitschaft für eingehende Messages ist deaktiviert worden.

6.13 StatusBtConnection

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

Ist eine Library-Funktion, die den (aktuellen) Status einer Bluetooth-Verbindung für den angegebenen Kanalindex zurückliefert, wenn die TransferAera nicht aktiv ist. Zurückgegeben werden die lokalen Informationen der Library über die Struktur BT_STATUS. Die Feldstärke wird dabei mit 0 angegeben. Die restlichen Statuswerte werden von der Library in Abhängigkeit von den gemeldeten Statusinformationen des Controllers gesetzt. Ist die TransferAera aktiviert, liefert diese Funktion für den angegebenen Kanalindex die aktuellen Statusinformationen inklusive der Feldstärke aus der TransferArea.

Aufruf-Parameter:

chanIdx	- Kanalindex (1 – 8)
BT_STATUS	- Pointer auf die Statusstruktur

Definition und Bedeutung der Statuswerte: siehe Kapitel 6.15 Statusanzeige in der TransferArea

6.14 Zusammenfassung der Statuswerte in den Callback-Funktionen

Auflistung der möglichen Statuswerte: (`enum CB_BtStatus`)

Status	Bedeutung
0	= **BT_SUCCESS** , Aktion erfolgreich
1	= **BT_CON_EXIST** , Bereits verbunden
2	= **BT_CON_SETUP** , Verbindungsaufbau zu dieser BT-Adresse wird bereits aktiv durchgeführt
3	= **BT_SWITCHED_OFF** , Verbindung fehlgeschlagen: Bluetooth ist lokal per Konfiguration abgeschaltet
4	= **BT_ALL_CHAN_BUSY** , Verbindung fehlgeschlagen: kein Bluetooth-Kanal lokal mehr frei
5	= **BT_NOT_ROBOTX** , Verbindung fehlgeschlagen: nicht verbindbares fremdes BT-Gerät (kein ROBO TX Controller)
6	= **BT_CON_TIMEOUT** , fehlgeschlagen: Timeout, kein Gerät unter dieser Adresse erreichbar (Timeout)
7	= **BT_CON_INVALID**, es existiert keine aktive Verbindung mit dem angegebenen Kanalindex.
8	= **BT_CON_RELEASE**, Verbindungsabbau zu dieser BT-Adresse ist bereits aktiviert und wird durchgeführt
9	**= BT_LISTEN_ACTIVE**, die Listen-Funktion ist für den angegebenen Kanalindex bereits aktiviert worden.
10	**= BT_RECEIVE_ACTIVE**, die Receive-Funktion ist bereits aktiviert worden.
11	**= BT_CON_INDICATION**, Signalisierung eines Verbindungsaufbaus auf der passiven Seite. Eine Bluetooth-Verbindung von der vorgegebenen Bluetooth-Adresse ist eingegangen.
12	**= BT_DISCON_INDICATION**, Signalisierung eines passiven Verbindungsabbaus (z.B. von der Gegenseite ausgelöst). Die Bluetooth-Verbindung besteht nicht mehr.
13	**= BT_MSG_INDICATION**, Signalisiert den Empfang einer Message von der Gegenstelle
14	**= BT_CHANNEL_BUSY**, der angegebene Kanalindex ist bereits registriert (Rufbereitschaft) bzw. in Verwendung (aktive Verbindung).
15	**= BT_BTADDR_BUSY**, zu dieser Bluetooth Adresse existiert bereits eine Rufbereitschaft oder eine aktive Verbindung über einen anderen Kanalindex.
16	**= BT_NO_LISTEN_ACTIVE**, auf der Gegenstelle ist keine Listen-Funktion aktiviert worden, kein Verbindungsaufbau möglich.

6.15 Statusanzeige in der TransferArea

Für jeden Kanalindex (1-8) gibt es in der TransferArea eine Connection-Status-Struktur, in der Informationen über den aktuellen Verbindungsstatus und die Funksignalstärke abgelegt werden. Beim Starten und Beenden der TransferArea werden die Werte mit 0 initialisiert.

Mögliche Bluetooth-Connection-Status Werte:

```
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
};
```

Datenstruktur:

```
typedef struct {             // 8 Byte
    UINT16  ConState;        // Status der Verbindung (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;
```

Eingebunden wird die Bluetooth Statusanzeige in der vorhandenen Struktur FTX1_STATE. Die Statusinformationen in der Struktur werden bei aktiver TransferArea jede Sekunde aktualisiert.

```
#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 Funktionen der I²C-API

Ein ROBO TX-Controller kann über seine I²C-Schnittstelle (Anschluss EXT2) externe Sensoren und Aktoren mit I²C-Schnittstelle betreiben. Diese können über die nachfolgend beschriebenen I²C -API-Funktionen von einem PC-Programm aus angesprochen werden.

Die I²C-Schnittstelle ist von ihrer Anschlussbelegung und interner Verdrahtung im einführenden Kapitel 7 (I²C-Schnittstelle) im Dokument „PC_Programmierung_RoboTXC.pdf“ Version 1.5 beschrieben.

7.1 Einführung

Jeder der nachfolgenden Funktionsaufrufe stellt einen eigenen I²C-Buszugriff unter Angabe der Geräteadresse (Device Address) dar. So ist es möglich eine Vielzahl von unterschiedlichen I²C-Geräten auch innerhalb eines Programms anzusprechen.

Wichtige Hinweise:
Die I²C-Geräteadresse (Device Address) ist laut I²C-Spezifikation mit nur 7 Bit anzugeben (Wertebereich: 0 – 127).
Weiterhin sind die I²C-Geräteadressen 80 und 84 (0x50 und 0x54) reserviert für ein internes EEPROM des ROBO TX-Controllers. Der Zugriff auf diese Adressen wird von der API nicht erlaubt. Die I²C-Geräteadressen 81-83 und 85-87 (0x51-0x53 und 0x55-0x57) sind ebenfalls von Speicherbereichen desselben EEPROMs belegt, werden aber von der Firmware nicht verwendet. Hier kann es (muss aber nicht) bei externen I2C-Geräten, die eine dieser Adressen verwenden, zu Buszugriffskonflikten kommen.

7.2 ftxI2cRead

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

Es wird auf dem I²C-Bus an der Geräteadresse „DevAddr“ und ggf. innerhalb des Gerätes an der Unteradresse „Offset“ ein Byte (8-Bit) oder zwei Bytes (16-Bit) gelesen. Mit dem Parameter „Flags“ werden die Adressierung, die Datenbusbreite, die Byte-Reihenfolge (nur bei 16-Bit), das Verhalten bei Busfehlern und die Zugriffsgeschwindigkeit festgelegt. Über die Callback-Funktion wird das Ergebnis des Lesezugriffs sowie das gelesene Datum asynchron zurückgegeben, sowie dieser Zustand eintritt.

Aufruf:	fthdl	- gültiger Handle des lokalen ROBO-TX Controllers
	BYTE DevAddr	- I2C Geräteadresse (Device Address)
	DWORD Offset	- wenn innerhalb des Gerätes eine Adressierung notwendig ist, dann wird diese interne Adresse hier übergeben. Der Wert von „Flags“ spezifiziert die Länge der internen Adresse in Bit 0..1.

BYTE Flags - Verwendete Zugriffsflaggen:

Bit 0..1	Adressierung	00: keine („Offset“ ungültig) 01: 8-Bit Adressierung 10: 16-Bit Adressierung MSB zuerst 11: 16-Bit Adressierung LSB zuerst
Bit 2..3	Datenbreite	00: - *nicht erlaubt* - 01: 8-Bit Daten (1 Byte) 10: 16-Bit Daten (2 Bytes) MSB zuerst 11: 16-Bit Daten (2 Bytes) LSB zuerst
Bit 4	KeepOpen	0: normaler Zugriff 1: schneller Zugriff ohne STOP/START
Bit 5..6	Error Mask = Verhalten bei Busfehler	00: Abbruch 01: Bis zu 10x wiederholen 10: Wiederholen bis Erfolg 11: - *nicht erlaubt* -
Bit 7	Taktfrequenz	0: standard (100 kHz) 1: fast (400 kHz)

cbFunc - Callback-Funktion, die das Ergebnis der Operation asynchron zurückmeldet.

Return:

DWORD errCode - FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code für einen Fehler beim Aufruf (siehe 7.4).

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur I2C_CB

Datenstruktur:

```
typedef struct {              //  4 bytes
      UINT16      value;      //  Gelesenes Datum (bei 8-Bit nur LSByte gültig)
      UINT16      status;     //  Ergebnis der I2C-Bus-Operation (siehe 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))*

Es wird auf dem I^2C-Bus an der Geräteadresse „DevAddr“ und ggf. innerhalb des Gerätes an der Unteradresse „Offset“ ein Byte (8-Bit) oder zwei Bytes (16-Bit) geschrieben. Mit dem Parameter „Flags“ werden die Adressierung, die Datenbreite, die Byte-Reihenfolge (nur bei 16-Bit), das Verhalten bei Busfehlern und die Zugriffsgeschwindigkeit festgelegt. Über die Callback-Funktion wird das Ergebnis des Schreibzugriffs asynchron zurückgegeben, sowie dieser Zustand eintritt.

Aufruf: fthdl - gültiger Handle des lokalen ROBO-TX Controllers

BYTE DevAddr - I2C Geräteadresse (Device Address)

DWORD Offset - wenn innerhalb des Gerätes eine Adressierung notwendig ist, dann wird diese interne Adresse hier übergeben. Der Wert von „Protocol“ spezifiziert die Länge der internen Adresse

WORD Data - Datum (8-Bit oder 16-Bit), das geschrieben werden soll

BYTE Flags - Verwendete Zugriffsflaggen:

Bit 0..1	Adressierung	00: keine („Offset“ ungültig) 01: 8-Bit Adressierung 10: 16-Bit Adressierung MSB zuerst 11: 16-Bit Adressierung LSB zuerst
Bit 2..3	Datenbreite	00: - *nicht erlaubt* - 01: 8-Bit Daten (1 Byte) 10: 16-Bit Daten (2 Bytes) MSB zuerst 11: 16-Bit Daten (2 Bytes) LSB zuerst
Bit 4	KeepOpen	0: normaler Zugriff 1: schneller Zugriff ohne STOP/START
Bit 5..6	Error Mask = Verhalten bei Busfehler	00: Abbruch 01: Bis zu 10x wiederholen 10: Wiederholen bis Erfolg 11: - *nicht erlaubt* -
Bit 7	Taktfrequenz	0: standard (100 kHz) 1: fast (400 kHz)

cbFunc - Callback-Funktion, die das Ergebnis der Operation asynchron zurückmeldet.

Return:

DWORD errCode - FTLIB_ERR_SUCCESS (kein Fehler) oder Error-Code für einen Fehler beim Aufruf (siehe 7.4).

Return-Werte der Callback-Funktion:

*data - Pointer auf Datenstruktur I2C_CB

Datenstruktur:

```
typedef struct {              //  4 bytes
      UINT16      value;      //  Geschriebenes Datum wiederholt
      UINT16      status;     //  Ergebnis der I2C-Bus-Operation (siehe 7.4)
} I2C_CB;
```

7.4 Fehler-Codes I²C-API-Funktionen (errCode bzw. status)

Return-Werte beim Aufruf (errCode)

Wert von errCode	Bedeutung
0x00000000	Funktionsaufruf erfolgreich FTLIB_ERR_SUCCESS
0xE0005000	Ungültige Device-Adresse FTLIB_I2C_INVALID_DEV_ADDR
0xE0005001	Ungültiges Flag für Adressbreite FTLIB_I2C_INVALID_FLAGS_ADDRMODE
0xE0005002	Ungültiges Flag für Datenbreite FTLIB_I2C_INVALID_FLAGS_DATAMODE
0xE0005003	Ungültiges Flag für Error Mask (Verhalten bei Busfehler) FTLIB_I2C_INVALID_FLAGS_ERRMODE

Return-Werte beim Callback (status)

Wert von status	Bedeutung
0	I2C-Operation erfolgreich ausgeführt I2C_SUCCESS
1	I2C-Lesefehler I2C_READ_ERROR
2	I2C-Schreibfehler I2C_WRITE_ERROR

8 Fehlercodes

Die Include-Datei ftErrCode.h enthält Fehlercodes:

```
#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 Dateien zum 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

//  Verbindung zum Controller öffnen
#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 Speicherlayout der TransferArea

Die TransferArea der Library "ftMscLib" gliedert sich in mehreren Datenstrukturen auf (siehe dazu auch Aufbau der TransferArea Kapitel 1 Allgemeines). Für den Abgleich der Output- und Input-Strukturen werden die Informationen, wie bei der „alten Library“ durch einen Kommunikationsthread, zum ROBO-TX Controller gesendet und empfangen. Der Zyklus für den Abgleich wird über einen MultiMedia-Timer gesteuert und beträgt 10ms. Die TransferArea innerhalb der Library besitzt den gleichen Aufbau wie die TransferArea innerhalb der Firmware auf dem Controller.

Im Folgenden werden hier lediglich die Strukturvariablen in den einzelnen Strukturen beschrieben, die für die Kommunikation mit dem ROBO-TX Controller oder zum Auslesen von bestimmten Information von Bedeutung sind. Alle anderen Felder werden durch die Firmware benutzt und können von der Library weder verändert, bzw. noch ausgelesen werden.

Hinweis: Alle hier nicht aufgeführten Definitionen, Datenstrukturen sind in den Header-Dateien

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

definiert.

9.1 Struktur FT_VERSION

Enthält Informationen über Versionen der Fimrware und der Hardware des ROBO-TX Controllers.

```
// 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	Datentyp	Bedeutung
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 der Transfer Area ("V %d.%02d", ta.part.c, ta.part.d)

9.2 Struktur TA_INFO

Enthält Informationen über den 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	Datentyp	Bedeutung
device_name	char[]	Speichert den Namen des ROBO-TX Controllers, kann über eine Library-Funktion geändert werden
bt_addr	char[]	Speichert die vergebene Bluetooth-Adresse
ta_array_start_addr	UINT32	Anfangsadresse des TransferAreaArray-Speicherbereichs
pgm_area_start_addr	UINT32	ProgrammArea Startadresse
pgm_area_size	UINT32	ProgrammArea Länge
version	FT_VERSION	Liefert die installierte Firmware- und Hardware-Version (siehe Struktur FT_VERSION)

9.3 Struktur BT_STATUS

Bestimmte Bluetooth Statusinformationen können gelesen werden.

```
// 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	Datentyp	Bedeutung
conn_state	UINT16	Siehe enum BtConnState
is_listen	BOOL16	Wenn TRUE - Bluetooth Channel wartet auf eingehende Verbindung (listening)
is_receive	BOOL16	Wenn TRUE - Bluetooth Channel ist bereit Nachrichten zu empfangen
link_quality	UINT16	0...31, 0 = schlechteste, 31 = beste Signalqualität

9.4 Struktur TA_STATE

Bestimmte Statusinformationen können gelesen und abgelegt werden. Für einen Konfigurationsabgleich z.B. wird die Variable *config_id* incrementiert und somit erkennt der Kommunikationsthread automatisch, dass die in der TranserArea vorliegende Konfiguration auf dem ROBO-TX Controller aktualisiert werden soll. Die Variable wird von der Library ständig überwacht.

```
// 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	Datentyp	Bedeutung
config_id	UINT8	Markiert eine Konfigurationsänderung (Inputs, Counter usw.) in der TransferArea, die Konfigurationsänderung wird mit dem nächsten Abgleich auf dem Controller aktiviert. Muss jedesmal um 1 erhöht werden, wenn etwas an der Konfiguration verändert wurde.
ext_dev_connect_state	BOOL8[]	Kennung, welche Slaves am RS485-Bus aktiv sind
btstatus	BT_STATUS[]	Status der Bluetooth-Verbindungen (siehe Struktur BT_STATUS)

9.5 Struktur TA_CONFIG

Die Struktur legt die Konfiguration der Motorausgänge, Universaleingänge und der Zähleingänge fest. Diese Struktur wird bei jeder Konfigurationsänderung zum ROBO-TX Controller gesendet und aktiviert damit die Konfiguration an den Ein- und Ausgängen.

```
// 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	Datentyp	Bedeutung
motor	BOOL8[]	Legt fest, ob die Ausgänge als 4 Motorausgänge (=TRUE) oder als 8 Digitalausgänge (=FALSE, PWM-Ausgang) definiert sind.
uni	UNI_CONFIG[]	Legt die Konfiguration der 8 Universaleingänge fest: mode= 0, digital= TRUE - Digital 10V (Spursensor) mode= 0, digital= FALSE - Analog 10V (Farbsensor) mode= 1, digital= TRUE - Digital 5kOhm (Taster) mode= 1, digital= FALS - Analog 5kOhm (NTC, …) mode= 3, digital= o.B. - Ultraschallsensor
cnt	CNT_CONFIG[]	Legt die Konfiguration des Zähleingangs fest: mode= 0 - normal (Änderung von 0 -> 1) mode= 1 - invers (Änderung von 1 -> 0)

9.6 Struktur TA_INPUT

Bei jedem zyklischen Abgleich wird die Struktur FTX1_INPUT vom ROBO-TX Controller zur Library übertragen. Diese enthält zum Zeitpunkt des Abgleich alle aktuellen Eingangswerte und interne Flags zur Motorsteuerung.

```
// 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	Datentyp	Bedeutung
uni	INT16[]	Enthält für jeden Universaleingang den von der Firmware ermittelten aktuellen Wert.
cnt_in	INT16[]	Aktuelle Konfiguration des Zähleingangs (normal, invers)
counter	INT16[]	Aktueller Zählerstand des Zählers
display_button_left	INT16[]	Zeitmessung während der Taster gedrückt bleibt. Der Zählerstand liefert die Zeit in 10 ms Intervallen, wo der Taster gedrückt bleibt, 0= Taster ist nicht gedrückt
display_button_right	INT16[]	s.o., wie linker Taster
cnt_resetted	BOOL16[]	Wird auf 1 gesetzt, wenn das zuletzt ausgelöste Zurücksetzen des Zählereingangs (Counter Reset) erfolgt ist.
motor_pos_reached	BOOL16[]	Flags zur Motorsteuerung, für jeden Motor wird mitgeteilt, ob das Ziel nach den Vorgaben erreicht wurde (Flag != 0)

9.7 Struktur TA_OUTPUT

Der Inhalt der Struktur TA_OUTPUT wird mit jedem zyklischen Abgleich von der Library zum ROBO-TX Controller übertragen und aktualisiert damit die vorgegebenen Werte an den Ausgängen des Controllers.

```
// 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	Datentyp	Bedeutung
cnt_reset_cmd_id	UINT16[]	Anforderung einen Zählereingang zurückzusetzen (Counter Reset). Muss jedesmal um 1 erhöht werden, wenn ein Zurücksetzen erforderlich ist.
master	UINT8[]	Bei einer aktiven Motorsteuerung wird für den Slave-Motor hier die Id vom Master-Motor festgelegt. Die MasterId ist dabei der Motorindex (0 bis3) + 1. Beispiel: Motorsynchron-Steuerung mit Motor 1 (Master, Index= 0) und Motor 3 (Slave, Index= 2). Inhalt: master[2] = 1
duty	INT16[]	Aktuelle Duty-Werte für die Motorausgänge, falls Motor aktiv geschaltet ist (Variable *motor* aus Struktur TA_CONFIG auf TRUE gesetzt ist), jeweils paarweise, die Differenz aus beiden Werten ergibt den realen Duty-Wert entsprechend der Drehrichtung (CW, CCW) duty[0] => M1- , duty[1] => M1+ duty[2] => M2- , duty[3] => M2+ usw. Im PWM-Betrieb (Digitalausgang) können die Ausgänge unabhängig voneinander betrieben werden. duty[0] => O1 duty[1] => O2 duty[2] => O3 usw.
distance	UINT16[]	Distanzwert (counter value). Die ist die Position bei einer aktiven Motorsteuerung an welcher der Motor stoppen soll. Erst wenn der Distanzwert != 0 ist, wird die Motorsteuerung aktiviert
motor_ex_cmd_id	UINT16[]	Muss immer dann um 1 erhöht werden, wenn die Einstellungen für die aktive Motorsteuerung (duty und/oder distance) geändert wurden.

9.8 Struktur TA_DISPLAY

Die Struktur TA_DISPLAY.DISPLAY_MSG dient dazu, generierte Textmeldungen aus einer Applikation auf dem Display des ROBO TX Controllers anzuzeigen. Die Textausgabe bleibt solange sichtbar, bis entweder das Textfeld mit 0 initialisiert wird oder durch Drücken einer der beiden Taster auf dem Controller, womit der Text dann gelöscht wird.

```
// 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	Datentyp	Bedeutung
id	UINT8	Message-Id. Muss jedesmal um 1 erhöht werden, wenn ein neues Message-Fenster angezeigt werden soll.
text	char[]	Character-Feld zur Aufnahme eines Textes (ASCII), der auf dem Display des ROBO TX Controllers angezeigt wird. Maximale Testlänge beträgt DISPL_MSG_LEN_MAX=98

9.9 Struktur TA_STATUS

Bestimmte TransferArea Statusinformationen können gelesen werden.

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

Variable	Datentyp	Bedeutung
status	UINT8	Status der TransferArea: 0 - Transfer Area ist nicht aktiv 1 - Transfer Area ist aktiv 2 - Transfer Area wird synchronisiert
iostatus	UINT8	Status der I/O Kommunikation: 0 - Remote I/O Request wurde gesendet 1 - Setzen Konfiguration wurde gesendet
ComErr	INT16	Systemfehlermeldung im Falle eines Fehlers mit dem COM Port

9.10 Struktur TA_CHANGE

Struktur dient dazu, Änderungen an den Universal- und Zählereingängen mitzuteilen. Falls irgendeine Änderung festgestellt wurde, wird die Variable *ChangeStatus* auf TRUE gesetzt, ist diese FALSE liegen keine aktuellen Änderungen vor. Damit kann eine Applikation optimal die anliegenden Werte anzeigen.

```
// 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	Datentyp	Bedeutung
ChangeStatus	UINT8	Globales Anzeigen einer Änderung bei den Universal-Eingängen, Zählereingänge oder Timer-Variablen. TRUE - es liegt eine Änderung vor FALSE - die relevanten Werte wurden in der TransferArea nicht verändert
ChangeUni	UINT8	Jedes gesetzte Bit innerhalb der UINT8-Variablen zeigt an, das für den repräsentativen Universal-Eingang eine Änderung des anliegenden Werts erfolgte. Bit 0 - Universal-Eingang I1 Bit 1 - Universal-Eingang I2 usw.
ChangeCntIn	UINT8	Flags zum Anzeigen einer Konfigurationsänderungen an den Zähleingängen Bit 0 - Zähleingang 1 Bit 1 - Zähleingang 2 usw.
ChangeCounter	UINT8	Flags zum Anzeigen einer Änderung des Zählerstandes Bit 0 - Zähler 0 Bit 1 - Zähler 2 usw.

9.11 Struktur TA_TIMER

In der neuen TransferArea werden wie in der alten, Timer-Variablen verwaltet. Diese haben die gleiche Bedeutung wie in der vorherigen Version und dienen für bestimmte Timeout-Variablen. Die Timer-Variablen werden über einen MultiMedia-Timer eventgesteuert aktualisiert.

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

10 Versionshistorie dieses Dokument

Version	Datum	Autor	Änderungen Bemerkungen
1.4.24	21.09.2009	Peter Classen	Erste vollständig dokumentierte Fassung
1.5.08	17.03.2011	Peter Duchemin	Ergänzung Funktionen Bluetooth Message API
1.5.11	24.04.2012	Peter Duchemin	Ergänzung Funktionen I^2C API