knobloch
electronic

Beschreibung der KeLib Bibliothek

Version FtLib/KeLib: 1.77
Stand: 05.06.2010

Knobloch GmbH
Weedgasse 14
55234 Erbes-Büdesheim

entwicklung@knobloch-gmbh.de
www.knobloch-gmbh.de

1 Inhaltsverzeichnis

2 Robo-Interface Programme

In das Robo-Interface können bis zu drei Programme per Download gespeichert werden. Programm 1 und Programm 2 werden dauerhaft in einem FLASH-Speicher abgelegt, ein drittes Programm kann in das RAM gespeichert werden. Der RAM-Speicher wird gelöscht, wenn ein Flash-Programm gestartet wird, sowie bei Stromausfall am Interface.

Die Auswahl des aktiven Programms erfolgt durch den Programmtaster. Wird dieser länger als 0,5 Sek. betätigt, kann das gewünschte Programm ausgewählt werden. Es leuchten nacheinander die beiden Programm-LEDs für Programm 1 / 2 auf, wenn in den jeweiligen Speicherplätzen ein Programm gespeichert ist. Ist ein Programm im RAM abgelegt, wird dieses durch die beiden leuchtenden LED´s angezeigt. Sind die Speicherplätze leer, kann das betreffende Programm nicht ausgeführt werden.

Zum Starten oder Stoppen des angezeigten Programms muß der Programm-Taster kurz betätigt werden (< 500ms). Das Auswählen, Starten und Stoppen von Programmen kann auch über die Schnittstellen (USB, seriell, Funk) erfolgen.

3 PC-Schnittstellen

3.1 Allgemeines

Die Auswahl der Schnittstellen erfolgt am Robo-Interface per Tastendruck. Nach dem Einschalten ist der "AutoScan"-Modus aktiv. Es werden die USB, serielle Schnittstelle und das Funkmodul (falls vorhanden) überprüft, ob Daten vorhanden sind. Erkennbar ist dieser Zustand an dem Aufleuchten der Schnittstellen-Leuchtdioden.

Sobald eine Schnittstelle Daten sendet, werden die anderen Schnittstellen gesperrt. Die aktive Schnittstelle blinkt, um die Datenkommunikation anzuzeigen. Wenn länger als 300ms keine Daten über die aktive Schnittstelle fließen, schaltet sich der AutoScan-Modus wieder ein.

Durch Drücken des "Port"-Tasters wird die nächste Betriebsart anhand nachfolgender Tabelle ausgewählt.

1.	AutoScan	USB - Seriell - Funk [1)]
2.	AutoScan	USB - Seriell
3.	USB	
4.	Seriell	
5.	IR-Direkt	

1)

Diese Betriebsart wird nur aktiviert, wenn das Funkmodul installiert ist.

Wird der Port-Taster länger als 3 Sekunden betätigt, schaltet sich das Interface in den "Intelligent-Interface Online-Modus". Die serielle Schnittstelle arbeitet dann mit den Parametern 9600,n,8,1. Zur Anzeige der Betriebsart blinkt die "COM" - Leuchtdiode sehr schnell. In dieser Betriebsart verhält sich das Interface wie ein Intelligent Interface im Online Modus. Es können jedoch keine Programme heruntergeladen werden. Durch einen Druck auf den Port-Taster wird wieder die AutoScan Betriebsart eingestellt.

3.2 Mehrere USB-Interfaces an einem Rechner

Um mehrere Interfaces am USB-Bus betreiben zu können, muss zunächst jedem Interface eine eigene Seriennummer zugewiesen werden. Standardmäßig werden alle Interfaces mit der gleichen Seriennummer ausgeliefert, um Probleme beim Austausch von Interfaces zu vermeiden. Das Windows-Betriebssystem unterscheidet gleiche Geräte anhand Ihrer Seriennummer. Für "jede" Seriennummer wird dann der entsprechende Treiber installiert. Dazu benötigt man unter Windows 2000 / XP Administratorrechte.

Daher werden standardmäßig alle ROBO Interfaces und ROBO I/O-Extensions mit der gleichen Seriennummer ausgeliefert. Solange nur ein Interface an einem Computer verwendet wird, gibt es keine Probleme. Der Rechner unterscheidet die Produkte durch ihren Namen (ROBO Interface, ROBO I/O-Extension und Robo RF-DataLink) und an deren jeweiliger Seriennummer. Daher können ein Robo-Interface und eine Robo I/O-Extension gleichzeitig an einem Rechner betrieben werden, ohne die Seriennummer zu ändern, da es sich um unterschiedliche Produkte handelt.

Wenn aber mehrere gleiche Produkte (z.B. Robo Interfaces) an einem Computer über USB betrieben werden sollen, muss vorher die Seriennummer der zusätzlich an den Rechner anzuschließenden Interfaces geändert werden, damit diese vom Computer unterschieden werden können.

Hinweis: Bei Anschluss über die serielle Schnittstelle an den Rechner muss keine Seriennummer geändert werden.

Das Interface hat daher zwei Seriennummern gespeichert. Über die Software kann eingestellt werden, ob die Standardseriennummer "1" oder die vom Hersteller einprogrammierte Geräteseriennummer "2" beim Einschalten des Gerätes aktiv ist.

Das Ändern der Seriennummer kann über die Software FtDiag.exe, RoboPro oder die Funktion GetFtDeviceSetting(), bzw. SetFtDeviceSetting() der FtLib geändert werden.

Zum Ändern der Seriennummer darf nur ein Produkt am USB angeschlossen sein, da sonst der Computer dieses nicht unterscheiden kann. In FtDiag.exe rufen Sie nach "SCAN USB" und dem Klick auf den Button "USB Device" das Menü Eigenschaften / Setup auf. In dieser Maske kann man die gewünschte Seriennummer, die nach dem nächsten Einschalten aktiv ist, einstellen.

Achtung: Wird die Seriennummer geändert, muss beim nächsten Einschalten des Interface eventuell der Windows-Treiber neu installiert werden. Dafür benötigt man unter Windows 2000 / XP aber Administratorrechte. Hat man diese nicht, kann man den Treiber nicht installieren und könnte daher auf das Interface über USB nicht mehr zugreifen. In diesem Fall kann man beim Einschalten des Interface den PROG-Taster gedrückt halten, bis der Lampentest ("Lichtorgel") beim Einschalten beendet ist. Das Interface verwendet dann die Seriennummer "1" und wird wieder vom installierten Treiber erkannt (um z.B. die Seriennummer auf erlaubte Werte zu ändern).
Beachten Sie bitte, dass es sich bei den auf den Produkten aufgedruckten Seriennummern um Hex-Zahlen handelt!

Auch wenn USB "theoretisch" bis zu 127 Geräte zulässt, hat sich in der Praxis gezeigt, dass unter Windows XP (mit SP1) auf einem 3 GHz Pentium 4 nur etwa 4-5 Produkte zuverlässig parallel betrieben werden können (d.h. die Aktualisierungszeit der TransferArea beträgt max. 10ms). Unter Windows 98 sind es bis zu 10 Geräte. Dies liegt

daran, dass die internen Windows-Treiber für die Motherboardhardware unter XP nicht für "Echtzeitanwendungen" optimiert sind. Es ist daher sinnvoller, IO-Extensions an die Robo-Interfaces anzuschließen, statt jedes Produkt einzeln an USB.

4 Library für Windows (FtLib)

Die Library unterstützt die fischertechnik USB-Produkte (serielle- und USB-Schnittstelle), sowie das Intelligent-Interface an der seriellen Schnittstelle.
Die Library steht in folgenden Versionen für das Microsoft Visual C++ Studio 6.0 zur Verfügung:

FtLib_Static_LIBCMT_Release.lib = Multithreaded Statisch (Linkeroption: /MT)
Projekteinstellungen auf C/C++ Karteikarte unter "Code Generation - Laufzeitbibliothek": Multithreaded

FtLib_Static_LIBCMTD_Debug.lib = Multithreaded Statisch Debug (Linkeroption: /MT d)
Projekteinstellungen auf C/C++ Karteikarte unter "Code Generation - Laufzeitbibliothek": Multithreaded Debug

FtLib_Static_MSVCRT_Release.lib = Multithreaded DLL (Linkeroption: /MD)
Projekteinstellungen auf C/C++ Karteikarte unter "Code Generation - Laufzeitbibliothek": Multithreaded DLL

FtLib_Static_MSVCRTD_Debug.lib = Multithreaded DLL Debug (Linkeroption: /MD d)
Projekteinstellungen auf C/C++ Karteikarte unter "Code Generation - Laufzeitbibliothek": Multithreaded DLL Debug

Die nachfolgenden Funktionen sind auch in der dynamischen Library FtLib.DLL enthalten.

4.1 Funktion für das Device Handling

4.1.1 DWORD GetLibVersion (void)

USB: ja
COM: ja
Diese Routine liefert die Versionsnummer der aktuellen Bibliothek.

Aufruf: ---
Return: Versionsnummer 0 0 x y (4 Byte, xy = Version X.Y)

4.1.2 DWORD InitFtLib (void)

USB: ja
COM: ja

Um die Datensteuerung für das Interface nutzen zu können, müssen Variablen initialisiert und Speicherbereiche angefordert werden. Dies erfolgt mit dieser Routine. Sie muß als erste Routine aufgerufen werden. Wir empfehlen den Aufruf direkt in der Initialisierungsroutine der Applikation (z.B. OnInitDialog() bei MFC-Applikationen). Die Funktion ist multithread fähig, sodass Sie auch zu einem späteren Zeitpunkt aufgerufen werden kann.

Vor dem Beenden der Applikation muß das Gegenstück zu dieser Routine "CloseFtLib(void)" aufgerufen werden, um reservierte Speicher wieder freizugeben.

Aufruf:	---	
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler beim Init
	ansonsten ein Fehlercode	(s. FtLib.h)
z.B.:	FTLIB_ERR_LIB_IS_INITIALIZED	Library ist schon initialisiert
	FTLIB_ERR_USB_NOT_SUPPORTET_FROM_OS	USB wird vom Betriebssystem nicht unterstützt (Windows-95 und Windows-NT)

4.1.3 DWORD CloseFtLib (void)

USB: ja
COM: ja

Diese Routine sollte beim Beenden der Applikation aufgerufen werden. Sie gibt alle Speicherbereiche wieder frei und löscht noch offene Handles.

Aufruf:	---	
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.1.4 DWORD IsFtLibInit (void)

USB: ja
COM: ja

Diese Routine liefert die Information, ob InitFtLib() schon ausgeführt wurde.

Aufruf:	---	
Return:	FTLIB_ERR_LIB_IS_INITIALIZED	Bereich schon angelegt
	FTLIB_ERR_LIB_IS_NOT_INITIALIZED	Bereich noch nicht angelegt

4.1.5 DWORD InitFtUsbDeviceList (void)

USB: ja
COM: nein
Diese Routine erstellt eine Liste der aktuell angeschlossenen Devices und speichert devicespezifische Daten ab. Die Devices werden nicht geöffnet. Es wird auch die Variable "Anzahl Device" (für GetNumFtDevice()) gesetzt. Das erste Device in der erzeugten Liste wird mit Index "0" adressiert.

Für jedes am USB - Anschluss gefundene RF Data-Link Modul wird als Device-Typ ein "FT_ROBO_RF_DATA_LINK" gespeichert. Dieser Typ kann mit GetFtDeviceTyp () abgefragt werden. Zusätzlich wird bei einem RF Data-Link Modul auf der eingestellten Frequenz über Funk nach Robo-Interfaces gesucht. Für jedes gefundene Robo-Interface wird direkt im Anschluss ein weiterer Eintrag erzeugt, als Typ wird "FT_ROBO_IF_OVER_RF" eingetragen. Durch dieses Verfahren können alle acht möglichen Interfaces über Funk angesprochen werden. Es kann jedoch immer nur ein Modul zur gleichen Zeit geöffnet werden. Bitte beachten Sie, dass diese Funktion für jedes gefundene RF-Data-Link Modul einige Sekunden benötigt um die möglichen Robo-Interfaces zu finden. Die Programmausführung kann dadurch einige Sekunden verzögert werden.
Wird anschließend ein Transfer-Thread auf den Eintrag des RF Data-Link-Moduls gestartet, dann startet die FtLib den TransferThread zum ersten gefundenen Robo-Interface.

Wichtig: Die erzeugte Liste ist nicht sortiert. Nach jedem Aufruf dieser Funktion können die gefundenen Geräte in einer anderen Reihenfolge angeordnet sein (insbesondere wenn in der Zwischenzeit neue Geräte angeschlossen werden). Lediglich bei einem RF-Data-Link Modul befinden sich im Anschluss daran die über dieses Modul erreichbaren Robo-Interfaces.

Sollten beim Aufruf noch Devices geöffnet sein, beendet sich diese Funktion mit einem Fehler.

Aufruf:	---	
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
z.B.:	FTLIB_ERR_SOME_DEVICES_ARE_OPEN	

4.1.6 UINT GetNumFtUsbDevice (void)

USB: ja
COM: nein
Diese Routine liefert die Anzahl der angeschlossenen fischertechnik USB-Devices. Diese Zahl wird beim Aufruf von InitFtUsbDeviceList() gesetzt.

Aufruf:	---	
Return:	UINT	Anzahl der gefundenen Devices

4.1.7 FT_HANDLE GetFtUsbDeviceHandle (UCHAR ucDevNr)

USB: ja
COM: nein

Diese Routine liefert einen Handle auf das gewünschte USB-Device in der Tabelle, welche von InitFtUsbDeviceList() erzeugt wird. Das erste Device wird mit Index "0" adressiert. Wichtig: Die mit InitFtUsbDeviceList erzeugte Liste ist nicht sortiert. Nach jedem Aufruf dieser Funktion können die gefundenen Geräte in einer anderen Reihenfolge angeordnet sein. Über diesen Handle erfolgt der Zugriff auf das Device oder dessen Daten.

Aufruf:	UCHAR ucDevNr	Index in die USB-Device Tabelle
Return:	FT_HANDLE	Handle des USB-Device Bei Fehlern wird NULL geliefert

4.1.8 FT_HANDLE GetFtUsbDeviceHandleSerialNr (DWORD dwSN, DWORD dwTyp)

USB: ja
COM: nein

Diese Routine liefert einen Handle auf das USB-Device, wenn das durch die Seriennummer spezifizierte Gerät in der durch InitFtUsbDeviceList() erzeugten Liste vorhanden ist. Da eine Seriennummer für die unterschiedlichen Geräte mehrfach vorkommen kann, spezifiziert die Variable "dwTyp" die Geräteklasse. Mit dem Typ-Parameter "FT_AUTO_TYPE" wird der Handle auf das erste gefundene Gerät mit der gewünschten Seriennummer übergeben.

Aufruf:	DWORD dwSN	Seriennummer des USB-Device
	DWORD dwTyp	Typ des zu öffnenden Devices
	FT_AUTO_TYPE	Liefere das erste Device mit dieser SN
	FT_ROBO_IF_USB	Robo Interface am USB-Port
	FT_ROBO_IO_EXTENSION	Robo I/O-Extension
	FT_ROBO_RF_DATA_LINK	Robo RF Data Link
Return:	FT_HANDLE	Handle des USB-Device Bei Fehlern wird NULL geliefert

4.1.9 DWORD OpenFtUsbDevice (FT_HANDLE hFt)

USB: ja
COM: nein

Diese Routine öffnet je nach DeviceTyp verschiedene Kanäle zum angeschlossenen Device. Hinweis: Der Thread der das Device öffnet, muß es auch wieder schließen!

Aufruf:	FT_HANDLE hFt	Handle des USB-Device
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.1.10 FT_HANDLE OpenFtCommDevice(DWORD dwPort, DWORD dwTyp, DWORD dwZyklus, DWORD *pdwError)

USB: nein
COM: ja
Diese Routine öffnet für den angegebenen COM-Kanal (COM1…COM4) eine Verbindung und liefert einen Handle darauf. Abhängig vom angegeben Device Typ wird automatisch die Schnittstelle auf 9600 oder 38400 Baud eingestellt.

Der Wert in der Variablen "dwZyklus" bestimmt das Abfrageverhalten und ist interfacespezifisch:

FT_INTELLIGENT_IF:
Der Wert bestimmt, nach wieviel normalen "Eingänge"-Abfragen" die beiden Analogeingänge EX und EY am Intelligent Interface abgefragt werden. Standard ist "FT_ANALOG_CYCLE".

FT_ROBO_IF_COM
Bei langsamen Rechnern ist es möglich, dass bei einem großen Datenaufkommen die Interface nicht mehr in der geforderten Zeit abgefragt werden können. Aus diesem Grunde kann man über diesen Parameter bestimmen, ob einige selten benutzte Werte übertragen werden.
0 = Dies entspricht der Standardabfrage (für langsamere Rechner). Die Analogwerte A1 und AV der I/O Extensions 1..3 werden nicht abgefragt.
>0 = Die Analogwerte A1 und AV der I/O Extensions 1..3 werden abgefragt.
Hinweis:
Erst ab der Robo Interface Firmware 1.45.0.3 und der FtLib-Version 0.41 ist diese Funktionalität integriert.

Falls ein Fehler beim Einrichten der Verbindung auftritt, wird der Fehlercode in einer DWORD-Variablen abgespeichert, auf die der Zeiger "pdwError" zeigt (NULL= speichern nicht gewünscht).

Die Angabe des Typs ist dahingehend wichtig, dass beim Kommunikationsthread mit den korrekten Abfragecodes gearbeitet wird. Der Parameter "FT_INTELLIGENT_IF_SLAVE" bewirkt beim Kommunikationsthread, dass das Extension-Modul wie ein SLAVE-1 behandelt wird.

Getrennt wird die Verbindung wieder mit den Funktionen " CloseAllFtDevices()" und "CloseFtDevice()".

Aufruf:	DWORD	dwPort	PORT_COM1…PORT_COM4
	DWORD	dwTyp	Typ des angeschlossenen Interface
		FT_INTELLIGENT_IF	Intelligent Interface (9600 bps)
		FT_INTELLIGENT_IF_SLAVE	Intelligent Interface mit Extension-Modul (9600)
		FT_ROBO_IF_IIM	Robo-Interface im Intelligent-If Modus (9600)
		FT_ROBO_IF_COM	Robo-Interface am COM Port (38400)
	DWORD	dwZyklus	(nur bei Intelligent Interface, z.B.: FT_ANALOG_CYCLE)
	DWORD*	pdwError	Zeiger auf eine Error Variable
			Konstantenwerte: (s. FtLib.h)

Return: Handle auf das Device (NULL = Fehler).
*pdwError enthält dann einen Error-Code (s. FtLib.h)

4.1.11 DWORD SetFtDeviceCommMode (FT_HANDLE hFt, DWORD dwMode, DWORD dwParameter, USHORT *puiValue)

USB: ja
COM: nein

Diese Routine stellt die Betriebsart der seriellen Schnittstelle im Interface ein. Nach dem Einschalten befindet sich die Schnittstelle im Normalbetrieb. In dieser Betriebsart kann das Interface im Onlinebetrieb gesteuert werden.
Im "Messagebetrieb" können Nachrichten von einem Interface in ein weiteres über die serielle Schnittstelle gesendet werden. Hierfür wird ein X-Kabel (gekreuzte sende- und Empfangsleitung) benötigt.

Die eingestellte Betriebsart bleibt solange erhalten, bis mit dieser Funktion eine neue Betriebsart eingestellt wird. Durch drücken des "Port"-Tasters am Interface wird wieder die IF_COM_ONLINE Betriebsart eingestellt, , wenn sich das Interface im AutoScan-Betrieb befindet.

Der aktuell eingestellte Zustand kann über "IF_COM_PARAMETER" abgefragt werden. Das Ergebnis wird in der Variablen, auf die "puiValue" zeigt gespeichert, wenn beim Aufruf der Wert von "puiValue" ungleich NULL ist..

Mode =	IF_COM_ONLINE:	Standardmodus einstellen Die Schnittstelle wird auf Standardmodus eingestellt (Parameter: 38400,n,8,1).
Mode =	IF_COM_MESSAGE:	Nachrichtenversand über serielle Schnittstelle
Mode =	IF_COM_PARAMETER	Betriebsart auslesen *puiValue (Lowbyte) = eingestellter Modus

Aufruf:	FT_HANDLE hFt	Handle des USB-Device
	DWORD dwMode	Betriebsart der seriellen Schnittstelle:
	IF_COM_ONLINE	0x01: Onlinebetrieb (38400,n,8,1)
	IF_COM_MESSAGE	0x03: Messagebetrieb (38400,n,8,1)
	IF_COM_PARAMETER	0x05: nichts verändern, nur aktuellen Zustand abfragen und bei *puiValue abspeichern
	USHORT puiValue	Zeiger auf eine USHORT-Variable (NULL, wenn nicht gewünscht) Hier wird bei dwMode = IF_COM_PARAMETER das Ergebnis gespeichert LOW-Byte = aktuelle Betriebsart (s. dwMode)

Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.1.12 DWORD CloseAllFtDevices ()

USB: ja
COM: ja
Diese Routine schließt alle Verbindungen für alle möglichen Devices.

Aufruf:	---	
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.1.13 DWORD CloseFtDevice (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine schließt alle Verbindungen für das angegebene Device. Falls der Kommunikationsthread noch läuft, wird er gestoppt.
Hinweis: Der Thread, der das Device öffnet, muss es auch wieder schließen!

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.1.14 DWORD GetFtDeviceTyp (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert den Typ des angeschlossenen Device für den angegebene Device-Handle.
Hinweis: Diese Daten werden von InitFtUsbDeviceList() aus dem Device gelesen und abgespeichert, bzw. von OpenFtCommDevice() gesetzt.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	DWORD Device Typ	
	NO_FT_DEVICE	kein Device angeschlossen / unbekannt
	FT_INTELLIGENT_IF	Intelligent Interface (serial)
	FT_INTELLIGENT_IF_SLAVE	Intelligent Interface mit Extension (serial)
	FT_ROBO_IF_IIM	FT-Robo Interface im Intelligent-Interface-Modus (serial)
	FT_ROBO_IF_USB	Robo Interface am USB-Port
	FT_ROBO_IF_COM	Robo Interface am COM-Port
		ansonsten Fehlercode (wenn Wert > FT_MAX_TYP_NUMBER, s. FtLib.h)

4.1.15 LPCSTR GetFtSerialNrStrg (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf den String mit der aktuellen Seriennummer im ASCII-Format, unter der sich das Device am Rechner angemeldet hat.

Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert. Handelt es sich um ein Produkt, das keine Seriennummern unterstützt (z.B. Intelligent Interface), wird als Seriennummer eine "1" in den String geschrieben.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	LPCSTR	Zeiger auf String wenn Zeiger = NULL, dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.16 DWORD GetFtSerialNr (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert die aktuelle Seriennummer als Long-Integer (DWORD), unter das sich das Device am Rechner angemeldet hat. Bei einem Fehler wird "0" geliefert, da es diese Seriennummer nicht gibt.

Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert. Handelt es sich um ein Produkt, das keine Seriennummern unterstützt (z.B. Intelligent Interface), wird als Seriennummer eine "1" gesetzt.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	DWORD	Seriennummer des Device wenn "0", dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.17 LPCSTR GetFtFirmwareStrg (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf den String mit der aktuellen Interface-Firmware im ASCII-Format.
Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	LPCSTR	Zeiger auf String wenn Zeiger = NULL, dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.18 DWORD GetFtFirmware (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert die Versionsnummer der Firmware des angeschlossenen Device als Integer (DWORD). Bei einem Fehler wird "0" geliefert, da es diese Firmwareversion nicht gibt.
Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	DWORD	Firmwareversion des Device im Format 3.2.1.0 (3=High Byte, 0=Low Byte) wenn Wert = "0", dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.19 LPCSTR GetFtManufacturerStrg (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf den Manufacturer-String im ASCII-Format.
Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	LPCSTR	Zeiger auf String wenn Zeiger = NULL, dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.20 LPCSTR GetFtShortNameStrg (UCHAR ucDevNr)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf den ShortName-String des Device im ASCII-Format.
Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	LPCSTR	Zeiger auf String wenn Zeiger = NULL, dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.21 LPCSTR GetFtLongNameStrg (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf den LongName-String des Device im ASCII-Format. Hinweis: Diese Daten werden von InitFtUsbDeviceList() oder OpenFtCommDevice() aus dem Device gelesen und abgespeichert.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	LPCSTR	Zeiger auf String wenn Zeiger = NULL, dann Fehler (Bereich geöffnet, korrekte DevNr?)

4.1.22 LPCSTR GetFtLibErrorString (DWORD dwErrorCode, DWORD dwTyp)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf einen String, der den in dwErrorCode übergebenen Fehlercode beschreibt. Beim Aufruf kann in dwTyp bestimmt werden, ob die Fehlerkonstante als String geliefert wird, oder eine Beschreibung des Fehlers in englisch.

Aufruf:

	DWORD	dwErrorCode	Fehlercode
	DWORD	dwTyp	Typ des zurückgegeben Strings 0 = Konstante als String 1 = englischer Fehlertext
Return:	LPCSTR		Zeiger auf String

4.1.23 DWORD GetFtDeviceSetting (FT_HANDLE hFt, FT_SETTING *pSet)

USB: ja
COM: ja
Diese Routine schreibt in die durch den Zeiger pSet adressierte Struktur die aktuellen Interfacedaten des ausgewählten Device. Die FT_SETTING-Struktur ist im Headerfile ftlib.h definiert (nur die mit RW gekennzeichneten Variablen können mit der Funktion SetFtDeviceSetting() geändert werden).

Aufruf:	FT_HANDLE hFt	Handle des Device
	FT_SETTING*	Zeiger auf eine FT_SETTING Struktur
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_INVALID_PARAM	Zeiger *pSet ist NULL

4.1.24 DWORD SetFtDeviceSetting (FT_HANDLE hFt, FT_SETTING *pSet)

USB: ja
COM: ja

Diese Routine schreibt die Interface Daten, aus der durch den Zeiger pSet adressierten Struktur, in das gewünschte Device. Die FT_SETTING-Struktur ist im Headerfile ftlib.h definiert (nur die mit RW gekennzeichneten Variablen können mit dieser Funktion geändert werden).

Aufruf:	FT_HANDLE hFt	Handle des Device
	FT_SETTING*	Zeiger auf eine FT_SETTING Struktur Ist der übergebene Wert NULL, wird keine Struktur verwendet. Innerhalb der Struktur müssen nicht benutzte Funktionen durch einen NULL-Zeiger deaktiviert werden.
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_INVALID_PARAM	Zeiger *pSet ist NULL

4.1.25 DWORD SetFtDistanceSensorMode (FT_HANDLE hFt, DWORD dwMode, DWORD dwTol1, DWORD dwTol2, DWORD dwLevel1, DWORD dwLevel2, DWORD dwRepeat1, DWORD dwRepeat2)

USB: ja
COM: ja

Diese Routine initialisiert den D1/D2 Eingang am Interface zum Anschluß des fischertechnik Distanzsensors oder zum Messen von Spannungen im Bereich 0-10 Volt.

Diese Parameter können auch mit der Funktion SetDeviceSetting() eingestellt werden.

Wichtiger Hinweis:
Da die Betriebsart der D1 / D2 Eingänge durch Software eingestellt werden kann, empfehlen wir an diese Anschlüsse keine Spannungen "direkt" einzuspeisen, um Beschädigungen des Interface bei Softwarefehlern zu vermeiden. Da die Eingänge hochohmig sind sollte ein Widerstand von ca.
220 Ohm - 470 Ohm direkt an die D1 / D2 Buchse angeschlossen werden (Reihenschaltung). Wir empfehlen erst "dahinter" die zu messende Spannung anzuschließen.

Aufruf:	FT_HANDLE hFt	Handle des Device
	DWORD dwMode	Betriebsart der Anschlüsse: IF_DS_INPUT_VOLTAGE = Eingänge messen Spannungen IF_DS_INPUT_ULTRASONIC = Eingänge für ft-Distanzsensor

Die nachfolgenden Parameter sind abhängig von der eingestellten Betriebsart.
Für dwMode = IF_DS_INPUT_DISTANCE gilt:

DWORD dwTol1	Toleranzbereich D1 (empfohlen: 20)
DWORD dwTol2	Toleranzbereich D2 (empfohlen: 20)
DWORD dwLevel1	Schwellwert D1
DWORD dwLevel2	Schwellwert D2
DWORD dwRepeat1	Wiederholungswert D1 (empfohlen: 3)
DWORD dwRepeat2	Wiederholungswert D2 (empfohlen: 3)

Der Distanzsensor arbeitet mit Infrarot - Licht und kann daher durch äußere Einflüsse gestört werden (z.B. IR-Handsender). Um diese Störungen auszuschließen, kann durch die Angabe des Wiederholungswertes festgelegt werden, wie oft der "gleiche" Messwert gemessen werden muß, bis dieser als gültig erkannt wird. Da die Messwerte nun von einer zur nächsten Messung etwas schwanken können, gibt es den Toleranzbereich. Sobald eine neue Messung beginnt, dürfen sich die nachfolgenden Messwerte innerhalb dieses "Fensters" verändern, ohne dass es zu einem Neustart der Messungen kommt. Der Schwellwert bestimmt den Level für die Auswertung als "digitale" Abstandssensoren. Unterhalb des Schwellwertes wird eine logische "0", darüber eine "1" gemeldet.
In der TransferArea werden die ermittelten Zustände von der Firmware in den Speicherstellen "Base+0x0C" (digital) und "Base+0x1C / Base+0x1E" (analog) gespeichert.

Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.2 Funktion für die Online-Kommunikation

Das Abfragen und Steuern der Ausgänge erfolgt durch eine "Transferarea" (Kommunikationsspeicherbereich). Dieser Bereich wird nach dem Starten des Kommunikationsthread (mittels StartFtTransferArea) alle 10ms mit dem Interface abgeglichen. Dabei spielt es keine Rolle, ob das Interface über USB, seriell oder per Funk an den PC angeschlossen ist. Auch spielt es keine Rolle, ob es sich um ein Intelligent-Interface, Robo-Interface oder eine IO-Extension handelt. Über den Devicehandle erkennt die Bibliothek, um welches Produkt es sich handelt und aktualisiert spätestens alle 10ms die vorhandenen Eingänge / Ausgänge des jeweiligen Interfacetyps. Falls mehrere Interfaces am Rechner angeschlossen sind, kann für jedes der angeschlossenen Produkte ein eigener Thread gestartet werden. Den Aufbau der Transferarea findet man im Anschluß an dieses Kapitel.

4.2.1 DWORD StartFtTransferArea (FT_HANDLE hFt, NOTIFICATION_EVENTS* sNEvent)

USB: ja
COM: ja

Diese Routine startet einen Kommunikationsthread, der kontinuierlich die Daten aus dem Transfer-Bereich mit dem Device abgleicht. Diese Funktionalität wird für den Online-Betrieb benötigt.

Das Device muß zuvor über die Funktion OpenFtUsbDevice() oder OpenFtCommDevice() geöffnet werden. Je nach Typ des seriellen Device (beim Öffnen der Schnittstelle mit OpenFtCommDevice() angegeben) werden nur die vom Interface unterstützten Werte abgefragt.

Beim Aufruf kann der Funktion ein Zeiger auf eine NOTIFICATION_EVENTS Struktur übergeben werden (NULL wenn nicht vorhanden). In dieser Struktur können Zeiger bzw. Handles für eine CallBack-Routine, Event- und Message-Handles übergeben werden. Immer wenn der Thread neue Daten vom Device erhält (spätestens alle 10ms) werden Nachrichten gesendet, bzw. die CallBack-Routine gerufen, wenn deren Zeiger- bzw. Handlewerte ungleich NULL sind. Die aufgerufenen CallBack-Routinen dürfen keine FtLib Funktionen aufrufen, da sonst ein Deadlock entstehen könnte.

Aufruf:	FT_HANDLE hFt	Handle des Device
	NOTIFICATION_EVENTS*	Zeiger auf eine Notification-Struktur Ist der übergebene Wert NULL, wird keine Struktur verwendet. Innerhalb der Struktur müssen nicht benutzte Funktionen durch einen NULL-Zeiger deaktiviert werden.
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.2.2 DWORD StartFtTransferAreaWithCommunication (FT_HANDLE hFt, NOTIFICATION_EVENTS* sNEvent)

USB: ja
COM: ja

Diese Routine startet einen Kommunikationsthread, der kontinuierlich die Daten aus dem Transfer-Bereich mit dem Device abgleicht. Diese Funktionalität wird für den Online-Betrieb benötigt. Zusätzlich werden Nachrichten zum Interface gesendet und empfangene Nachrichten abgeholt. Die FtLib besitzt einen Nachrichtenzwischen-speicher, um die durch SendFtMessage() erhaltenen Nachrichten zwischen- zuspeichern. StartFtTransferAreaWithCommunication() sorgt nun dafür, dass diese Nachrichten innerhalb der Onlineabfrage an das Interface gesendet werden.

Das Device muss zuvor über die Funktion OpenFtUsbDevice() oder OpenFtCommDevice() geöffnet werden. Je nach Typ des seriellen Device (beim Öffnen der Schnittstelle mit OpenFtCommDevice() angegeben) werden nur die vom Interface unterstützten Werte abgefragt. Wenn Nachrichten über die serielle Schnittstelle gesendet werden sollen, muss diese zuvor durch die Funktion SetFtDeviceCommMode() auf die Betriebsart IF_COM_MESSAGE eingestellt werden.

Beim Aufruf kann der Funktion ein Zeiger auf eine NOTIFICATION_EVENTS Struktur übergeben werden (NULL wenn nicht vorhanden). In dieser Struktur können Zeiger bzw. Handles für eine CallBack-Routine, Event- und Message-Handles übergeben werden. Immer wenn der Thread neue Daten vom Device erhält (spätestens alle 10ms) werden Nachrichten gesendet, bzw. die CallBack-Routine gerufen, wenn deren Zeiger- bzw. Handlewerte ungleich NULL sind. Die aufgerufenen CallBack-Routinen dürfen keine FtLib Funktionen aufrufen, da sonst ein Deadlock entstehen könnte.

Wird vom Interface während der Online-Abfrage eine Nachricht erhalten, wird die in der Variablen "CallbackMessage" gespeicherte Adresse der Callback-Funktion aufgerufen und ein Zeiger auf die Nachricht übergeben. Es liegt dann in der Verantwortung des Programmierers, diese Nachricht schnellstmöglich für das Hauptprogramm zwischenzuspeichern, um die Ausführungszeit der Onlineabfrage nicht zu verzögern.

Aufruf:	FT_HANDLE hFt	Handle des Device
	NOTIFICATION_EVENTS*	Zeiger auf eine Notification-Struktur Ist der übergebene Wert NULL, wird keine Struktur verwendet. Innerhalb der Struktur müssen nicht benutzte Funktionen durch einen NULL-Zeiger deaktiviert werden.
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.2.3 DWORD StopFtTransferArea (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine stoppt den Kommunikationsthread des gewünschten Device.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.2.4 FT_TRANSFER_AREA* GetFtTransferAreaAddress (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine liefert einen Zeiger auf die Adresse der Transfer Area.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	FT_TRANSFER_AREA*	Zeiger auf Struktur wenn NULL, dann Fehler

4.2.5 DWORD IsFtTransferActiv (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine überprüft, ob am angegebenen Device ein Kommunikationsthread aktiv ist.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:		
FTLIB_ERR_THREAD_IS_RUNNING		wenn ein Thread aktiv ist
FTLIB_ERR_THREAD_NOT_RUNNING		wenn kein Thread aktiv ist Es kann auch ein Fehlercode geliefert werden, wenn dies erforderlich ist. (s. FtLib.h)

4.2.6 DWORD ResetFtTransfer (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine löscht alle Ausgänge am angegebenen Device, sowie an allen angeschlossenen I/O-Extensions (bzw. Intelligent-Interfaces und Extension Modulen), sofern kein Kommunikationsthread aktiv ist.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.3 Funktion für die Nachrichtenverarbeitung

Mehrere Robo Interfaces können über die RS232-Schnittstelle und die Funkschnittstelle untereinander Nachrichten austauschen. Die Nachrichten bestehen grundsätzlich aus einem 16-Bit Nachrichten-ID und einem 16-Bit Wert.

Die Kommunikation erfolgt auf Funk-Kanälen (Frequenzen). Es können maximal acht Funk-Interfaces auf dem gleichen Funk-Kanal Nachrichten austauschen. Jedes Modul hat dabei eine eigene "ID" Nummer (RoboPro: "Funkrufnummer"), die im Bereich 1..8 eingestellt werden kann. Das PC-Modul hat immer die ID Nummer "0". Zusätzlich können Nachrichten auch an die serielle Schnittstelle des Interface gesendet und von dieser empfangen werden.

Jeder Funk-Kanal ist in 256 logische Unterkanäle unterteilt, die zur Strukturierung der Kommunikation dienen. Eine Nachricht wird dabei immer an alle Teilnehmer verteilt ("Broadcasting"). Jede Nachricht besteht aus fünf Datenbytes:

SubId	Nummer des Unterkanals (1 Byte)
Nachricht	Nachrichtendaten (4 Bytes)
	B1:B0: Nachrichten ID (Low-Word)
	B3:B2: Nachricht (High-Word)

Es gibt keine Rückmeldung an den "Absender", ob die Nachricht empfangen wurde.

Gesteuert wird die Kommunikation vom PC-Modul, das als Message-Router arbeitet.

Um die Nachrichtenverwaltung (Messagesystem) im Onlinebetrieb nutzen zu können, muss die Interfaceabfrage mit StartFtTransferAreaWithCommunication() gestartet werden. Das System ist ausgelegt, dass ca. alle 10ms zwei Nachrichten zum Interface übertragen werden und empfangene Nachrichten geholt werden.

4.3.1 Serielle Nachrichten

Im Online-Betrieb können Nachrichten auch über die serielle Schnittstelle gesendet und empfangen werden. Dazu muss die Betriebsart der Schnittstelle im Interface mit der Funktion "SetFtDeviceCommMode()" geändert werden. Die geänderte Betriebsart ist dann durch das dauerhafte Leuchten der COM-Leuchtdiode am Interface erkennbar. Durch einen Tastendruck auf den Port-Taster wird die Betriebsart wieder zurückgesetzt, selbstverständlich kann dies auch durch die Software erfolgen. Es ist also möglich, zwischen zwei Robo-Interfaces Nachrichten über die serielle Schnittstelle auszutauschen. Das dazu benötigte Verbindungskabel ("X-Kabel" mit gekreuzten Sende- und Empfangsleitungen) kann über den fischertechnik-Einzelteilservice bezogen werden.

4.3.2 FtLib Funktionen

Die FtLib bietet die Funktion "SendFtMessage ()" für den Datentransport an. Beim Senden kann über eine Hardware-Id ein physikalischer Kanal ausgewählt werden. Falls kein zweites Interface oder keine Funkschnittstelle zur Verfügung stehen, kann man die Nachricht "an sich selbst" senden. Man kann die Nachricht an die serielle Schnittstelle senden oder über Funk (RF) an andere Module durch den Message-Router verteilen lassen.

4.3.3 Nachrichtenempfang

Sobald das Interface eine Nachricht empfängt, ruft es eine Callback-Routine auf und übergibt dieser einen Zeiger auf die empfangene Nachricht. Die Adresse der Callback-Routine muss beim Starten des Transfers angegeben werden. Wir empfehlen, dass diese Callback-Routine die Nachricht in einen eigenen Puffer kopiert und vom Hauptprogramm zu einem späteren Zeitpunkt bearbeitet wird, um die Thread-Routine nicht zu lange "anzuhalten".

4.3.4 DWORD SendFtMessage (FT_HANDLE hFt, BYTE bHwId, BYTE bSubId, DWORD dwMessage, DWORD dwWaitTime DWORD)

USB: ja
COM: ja

Diese Routine schreibt eine Nachricht in den internen Puffer. Über den Parameter bHwId wird ein physikalischer Kanal ausgewählt, über den Parameter bSubId kann dieser Kanal in logische Unterkanäle aufgeteilt werden. Die eigentliche Nachricht besteht aus einem 16-Bit Nachrichten ID im Lowword und aus einem 16-Bit Nachrichtenwert im Highword der Variablen dwMessage.
Normalerweise schreibt diese Funktion die übergebene Nachricht in den internen Nachrichtenpuffer und kehrt umgehend zurück. Falls der interne Puffer jedoch voll ist, kann mit dem Parameter dwWaitTime eine Wartezeit in "ms" angegeben werden. Kann die Nachricht in der angegebenen Zeit nicht in den Puffer geschrieben werden, wird diese Funktion mit einer Fehlermeldung beendet. Bei dwWaitTime=0 wird nicht gewartet. Ist der Puffer voll, kehrt diese Funktion umgehend zurück.

Das Messagesystem benötigt für die Verteilung einer Nachricht etwa 5ms. Da insbesondere innerhalb Schleifen diese Funktion wesentlich öfters aufgerufen werden könnte, kann über den Parameter "dwOption" die Anzahl der zu sendenden Nachrichten optimiert werden. Dadurch kann vermieden werden, dass mehrfach die gleiche Nachricht gesendet wird.

Sollen auch Nachrichten an die serielle Schnittstelle des Interface gesendet werden (bHwId = MSG_HWID_SER), so muß die Betriebsart IF_COM_MESSAGE vor dem Starten des Transfer-Threads mit der Funktion " SetFtDeviceCommMode()" aktiviert werden.

Aufruf:	FT_HANDLE hFt	Handle des Device
	BYTE bHwId	Hardware-ID
	MSG_HWID_SELF (0x00):	direkt in den eigenen Receive-Puffer kopiert
	MSG_HWID_SER (0x01):	Über Interface-RS232 gesendet
	MSG_HWID_RF (0x02):	Über Funk nur an andere Module
	MSG_HWID_RF_SELF (0x03):	Über Funk auch an sich selbst
	BYTE bSubId	Logischer Kanal, erlaubt sind die ID´s 0…219. Die Werte von 220 bis 255 sind für interne Aufgaben reserviert
	DWORD dwMessage	Lowword: 16-Bit Nachrichten-ID Highword: 16-Bit Nachricht
	DWORD dwWaitTime	Falls der interne Puffer voll ist, kann mit diesem Parameter (in ms) bestimmt werden, wie lange gewartet wird, bis die Funktion wieder zurückkehrt.
	DWORD dwOption	Sendeoptionen
	MSG_SEND_NORMAL (0):	Die Nachricht wird direkt in den Sendepuffer geschrieben
	MSG_SEND_OTHER_THAN_LAST (1):	Die Nachricht wird nicht gesendet, wenn eine identische Nachricht (bHwId, bSubId, dwMessage) am Ende des Puffers steht. Ist der Puffer leer, oder es steht eine andere Nachricht am Ende des Puffers, wird die Nachricht gesendet.
	MSG_SEND_IF_NOT_PRESENT (2):	Die Nachricht wird nicht gesendet, wenn eine identische Nachricht (bHwId, bSubId, dwMessage) irgendwo im Puffer steht. Ist der Puffer leer, wird die Nachricht gesendet.

Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_MSG_BUFFER_FULL_TIMEOUT	Puffer ist voll, Nachricht konnte in der angegebenen Zeit nicht übertragen werden

4.3.5 DWORD ClearFtMessageBuffer (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine löscht die noch im internen Nachrichtenpuffer vorhandenen Nachrichten für das angegebene Device.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.4 Funktion für den Datendownload / Programmsteuerung

4.4.1 DWORD GetFtMemoryLayout (FT_HANDLE hFt, BYTE * pbArray, DWORD dwSize)

USB: ja
COM: ja
Diese Routine schreibt in das übergebene Array das aktuelle MemoryLayout des angeschlossenen Device. Zum Auslesen darf kein Kommunikationsthread aktiv sein.

Vor einem Programm-Download muss das Anwenderprogramm auf diese Adressen gelinkt werden.

Hinweis: Wir behalten uns vor, bei zukünftigen Firmwareversionen das Speicherlayout zu verändern, wenn dies erforderlich wird.

Aufruf:	FT_HANDLE hFt	Handle des Device
	BYTE * pbArray	Zeiger auf ein Byte-Array
	DWORD dwSize	Größe des Speicherbereichs in Byte
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_INVALID_PARAM	Zeiger *pbArray ist NULL

Aufbau des Memory-Layout

FLASH-1 Bereich Start – End Adresse (20 Bit: a bb cc – d ee ff)

Byte 1:	[0]	cc	(der Start Adresse)
Byte 2:	[1]	bb	(der Start Adresse)
Byte 3:	[2]	0a	(der Start Adresse)
Byte 4:	[3]	ff	(der End Adresse)
Byte 5:	[4]	ee	(der End Adresse)
Byte 6:	[5]	0d	(der End Adresse)

FLASH-2 Bereich Start – End Adresse (20 Bit: a bb cc – d ee ff)

Byte 7:	[6]	cc	(der Start Adresse)
Byte 8:	[7]	bb	(der Start Adresse)
Byte 9:	[8]	0a	(der Start Adresse)
Byte 10:	[9]	ff	(der End Adresse)
Byte 11:	[10]	ee	(der End Adresse)
Byte 12:	[11]	0d	(der End Adresse)

RAM Bereich Start – End Adresse (20 Bit: a bb cc – d ee ff)
(Nutzbar für Programme und Variablen, Bereich steht nach Start eines Flash-Programms dem Programm komplett zur Verfügung)

Byte 13:	[12]	cc	(der Start Adresse)
Byte 14:	[13]	bb	(der Start Adresse)
Byte 15:	[14]	0a	(der Start Adresse)
Byte 16:	[15]	ff	(der End Adresse)
Byte 17:	[16]	ee	(der End Adresse)
Byte 18:	[17]	0d	(der End Adresse)

Interner RAM1 Start – End Adresse (20 Bit: a bb cc – d ee ff)
(Nutzbar für Variablen und Stacks)

Byte 19:	[18]	cc	(der Start Adresse)
Byte 20:	[19]	bb	(der Start Adresse)
Byte 21:	[20]	0a	(der Start Adresse)
Byte 22:	[21]	ff	(der End Adresse)
Byte 23:	[22]	ee	(der End Adresse)
Byte 24:	[23]	0d	(der End Adresse)

Interner RAM2 Start – End Adresse (20 Bit: a bb cc – d ee ff)
(Nutzbar für Variablen und Stacks, bitweise Adressierung möglich)

Byte 25:	[24]	cc	(der Start Adresse)
Byte 26:	[25]	bb	(der Start Adresse)
Byte 27:	[26]	0a	(der Start Adresse)
Byte 28:	[27]	ff	(der End Adresse)
Byte 29:	[28]	ee	(der End Adresse)
Byte 30:	[29]	0d	(der End Adresse)

4.4.2 DWORD DownloadFtProgram (FT_HANDLE hFt, DWORD dwMemBlock, BYTE* pbArray, DWORD dwSize, DWORD dwParameter, BYTE *pbName, DWORD dwNameLen)

USB: ja
COM: ja
Diese Routine schreibt das übergebene Programm in den gewünschten Speicherbereich (MemBlock) des Interface. Das Programmieren dauert je nach Speichergröße einige Sekunden.
In der Variablen dwParameter kann angegeben werden, ob das Programm beim Einschalten automatisch gestartet werden soll oder nicht. Dies funktioniert jedoch nur bei Programmen für den Speicherbereich "Flash1". Bei allen anderen Speicherbereichen wird die Angabe ignoriert.

Beim Aufruf kann ein Zeiger auf eine Zeichenfolge, sowie dessen Länge (in Anzahl Byte) übergeben werden ("\0" am Ende des "String" daher nicht nötig). Diese Zeichenfolge wird als Programmname im gleichen Speicherblock abgespeichert. Da eine Überprüfung der Zeichenfolge auf "\0" nicht stattfindet, können auch Binärzahlen gespeichert werden. Falls der Zeiger "pbName" beim Aufruf den Wert "NULL" hat, oder dwNameLen eine "0" enthält, wird kein Name gespeichert. Die Funktion "GetFtProgramName()" liefert dann eine Zeichenfolge, die im ersten Byte ein Leerzeichen (0x20) und als restliche Zeichen "0" enthält.

Wichtiger Hinweis:
Wenn Sie diese Funktion benützen, müssen Sie wissen "was" Sie tun! Wenn das Programm gestartet wird, finden keine Sicherheitskontrollen mehr statt! Es ist daher möglich, durch Umprogrammieren der Hardwareregister im Prozessor auch Hardwareschäden zu erzeugen!!! Der Hersteller übernimmt keine Garantieleistungen wenn ein Defekt am Interface durch falsche Programme entstanden ist!

Ist das Programm mit dem Kennzeichen "Autostart" versehen, kann der Autostart durch Drücken des "Prog"-Tasters beim Einschalten übergangen werden. Der Taster muß spätestens während des LED-Tests betätigt werden. Sobald die LED´s der USB / COM-Schnittstellen den normalen Betrieb anzeigen, kann der Taster losgelassen werden.

Hinweis: Das Löschen der Flashspeicher kann je Block bis zu 15 Sekunden dauern! Dies muss bei einer TimeOut Überwachung berücksichtigt werden. Hinzu kommt dann die Zeit zum Übertragen und Speichern der Daten im Interface.
Richtwerte seriell:
1 k Datenübertragung in den Flash < 1 Sekunde
128k Datenübertragung in den Flash ca. 50 Sekunden

Richtwerte USB:
1 k Datenübertragung in den Flash < 1 Sekunde
128k Datenübertragung in den Flash ca. 8 Sekunden

Aufruf: DWORD DownloadFtProgram (FT_HANDLE hFt, DWORD dwMemBlock,
BYTE* pbArray, DWORD dwSize,
DWORD dwParameter, BYTE *pbName,
DWORD dwNameLen)

FT_HANDLE hFt		Handle des Device
	DWORD dwMemBlock	Nummer des Speicherbereiches 0 = Flash1 1 = Flash2 2 = RAM
	BYTE * pbArray	Zeiger auf ein Byte-Array
	DWORD dwSize	Anzahl der zu programmieren Byte
	DWORD dwParameter	Parameter: 0 = normal 1 = Autostart des Programms
	BYTE * pbName	Zeiger auf ein Byte-Array mit dem Programmnamen (max. 80 Zeichen) "NULL" = keinen Namen speichern
	DWORD dwNameLen	Länge des Programmnamens in Byte 0 = keinen Namen speichern
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Programm gespeichert
	FTLIB_INFO_PROGRAM_0_IS_RUNNING	Flash-1 Programm läuft
	FTLIB_INFO_PROGRAM_1_IS_RUNNING	Flash-2 Programm läuft
	FTLIB_INFO_PROGRAM_2_IS_RUNNING	RAM-Programm läuft

4.4.3 DWORD StartFtProgram (FT_HANDLE hFt, DWORD dwMemBlock)

USB: ja
COM: ja
Diese Routine startet das gewünschte Programm (0..2).

Aufruf:	FT_HANDLE hFt	Handle des Device
	DWORD dwMemBlock	Nummer des Speicherbereiches dessen Programm gestartet werden soll
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Programm gestartet
	FTLIB_INFO_PROGRAM_0_IS_RUNNING	Flash-1 Programm läuft
	FTLIB_INFO_PROGRAM_1_IS_RUNNING	Flash-2 Programm läuft
	FTLIB_INFO_PROGRAM_2_IS_RUNNING	RAM-Programm läuft
	FTLIB_ERR_IF_NO_PROGRAM	kein Programm gespeichert

4.4.4 DWORD StopFtProgram (FT_HANDLE hFt)

USB: ja
COM: ja
Diese Routine stoppt das aktuell laufende Programm.

Aufruf:	FT_HANDLE hFt	Handle des Device
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Programm wurde gestoppt
	FTLIB_ERR_IF_NO_PROGRAM_IS_RUNNING	

4.4.5 DWORD DeleteFtProgram (FT_HANDLE hFt, DWORD dwMemBlock)

USB: ja
COM: ja
Diese Routine löscht das angegebene Programm.
Hinweis: Das Löschen der Flashspeicher kann je Block bis zu 15 Sekunden dauern! Dies muss bei einer TimeOut Überwachung berücksichtigt werden.
Es findet keine Überprüfung statt, ob der Block ein Programm enthält. Es wird in jedem Fall gelöscht!

Aufruf:	FT_HANDLE hFt	Handle des Device
	DWORD dwMemBlock	Nummer des Speicherbereiches dessen Programm gelöscht werden soll
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Programm wurde gelöscht

4.4.6 DWORD SetFtProgramActiv (FT_HANDLE hFt, DWORD dwMemBlock)

USB: ja
COM: ja
Diese Routine aktiviert das angegebene Programm. Es werden also die Programm-LED´s eingeschaltet, ohne das Programm zu starten. Es ist auch möglich, die Programm-LED´s hiermit auszuschalten.

Aufruf:	FT_HANDLE hFt	Handle des Device
	DWORD dwMemBlock	Nummer des Speicherbereiches der "aktiviert" werden soll. Mit "-1" werden die LED´s abgeschaltet.
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Programm gestartet
	FTLIB_INFO_PROGRAM_0_IS_RUNNING	Flash-1 Programm läuft
	FTLIB_INFO_PROGRAM_1_IS_RUNNING	Flash-2 Programm läuft
	FTLIB_INFO_PROGRAM_2_IS_RUNNING	RAM-Programm läuft
	FTLIB_ERR_IF_NO_PROGRAM	kein Programm gespeichert
	FTLIB_ERR_DOWNLOAD_CRC	CRC-Fehler
	FTLIB_ERR_POWER_TOO_LOW	Spannung am IF zu niedrig

4.4.7 DWORD GetFtProgramName (FT_HANDLE hFt, DWORD dwMemBlock, DWORD dwSize, LPVOID pName)

USB: ja
COM: ja
Diese Routine ließt den Namen des angegebenen Programms aus. Beim Aufruf muß ein Zeiger auf einen entsprechend großen Datenblock übergeben werden, in den der Zeichenstring kopiert wird.

Tipp:
Mit dieser Funktion kann auch überprüft werden, ob ein Programm im angegebenen Speicherbereich gespeichert ist. Dabei spielt es keine Rolle, ob beim Abspeichern ein Name angegeben wurde oder nicht. Ist kein Programm gespeichert, wird der Fehlercode FTLIB_ERR_IF_NO_PROGRAM geliefert.

Aufruf:	FT_HANDLE hFt	Handle des Device
	DWORD dwMemBlock	Nummer des Speicherbereiches
	DWORD dwSize	Größe des Speichers (mind. 80 Zeichen)
	LPCSTR pName*	Zeiger auf den Speicherbereich für Prog.Name
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Programm gestartet
	FTLIB_INFO_PROGRAM_0_IS_RUNNING	Flash-1 Programm läuft
	FTLIB_INFO_PROGRAM_1_IS_RUNNING	Flash-2 Programm läuft
	FTLIB_INFO_PROGRAM_2_IS_RUNNING	RAM-Programm läuft
	FTLIB_ERR_IF_NO_PROGRAM	kein Programm gespeichert

4.4.8 DWORD GetFtMemoryData (FT_HANDLE hFt, BYTE* pbArray, DWORD dwSize, DWORD dwAddress)

USB: ja
COM: ja

Diese Routine ließt 64 Datenbytes aus dem Speicher des Interface ab der in dwAddress angegebenen Adresse und schreibt die Daten in das Array, auf das pbArray zeigt. Das Array muß mindestens 64 Bytes groß sein (dwSize). Es können nur die Speicherbereiche von 0x00400 bis 0xDFFFF abgefragt werden. Bei Speicherbereichen außerhalb dieses Bereiches wird eine "00" geliefert.

Hinweis: Diese Funktion arbeitet auch bei einem laufenden Programm ("Aktiv" Modus) Das Lesen von Daten über die serielle Schnittstelle ist nur möglich, wenn kein Kommunikationsthread läuft. Bitte beachten Sie, dass der Kommunikationsbereich im Interface ab der Adresse 0x400 nur von laufenden Programmen (Aktiv-Modus) im Interface genutzt wird. Im Online-Modus (Kommunikationsthread) ist dieser Bereich deaktiviert.

Aufruf:	FT_HANDLE hFt	Handle des Device
	BYTE * pbArray	Zeiger auf ein Byte-Array
	DWORD dwSize	Größe des Byte-Array
	DWORD dwAddress	Adresse im Interface
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)
	gängige Fehlercodes:	
	FTLIB_ERR_SUCCESS	Daten gelesen
	FTLIB_ERR_THREAD_IS_RUNNING	Nur COM: Kommunikationsthread läuft
	FTLIB_ERR_INVALID_PARAM	Zeiger pbArray ist NULL

4.4.9 DWORD WriteFtMemoryData (FT_HANDLE hFt, DWORD dwData, DWORD dwAddress)

USB: ja
COM: ja

Diese Routine schreibt ein Datenbyte in den Speicher des Interface ab der in dwAddress angegebenen Adresse. Es können nur die Speicherbereiche von 0x00400 bis 0x7FFFF beschrieben werden. Nach dem Schreiben wird ein Verify durchgeführt.

Hinweis: Diese Funktion arbeitet auch bei einem laufenden Programm ("Aktiv" Modus) Das Schreiben von Daten über die serielle Schnittstelle ist nur möglich, wenn kein Kommunikationsthread läuft. Bitte beachten Sie, dass der Kommunikationsbereich im Interface ab der Adresse 0x400 nur von laufenden Programmen (Aktiv-Modus) im Interface genutzt wird. Im Online-Modus (Kommunikationsthread) ist dieser Bereich deaktiviert.

Aufruf:	FT_HANDLE hFt	Handle des Device
	DWORD dwData	zu schreibendes Datenbyte
	DWORD dwAddress	Adresse im Interface
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler
	ansonsten ein Fehlercode	(s. FtLib.h)

4.5 education line

Das Setzen und Abfragen der IO´s erfolgt nicht über eine Transfer-Area wie beim Robo-Interface, sondern über eine IO-Struktur, da hier keine zeitkritischen Dinge passieren. Für die Steuerung der education line Modelle werden nur wenige Datenbytes benötigt. Im USB-Bereich spielt es jedoch keine Rolle, ob 1 Byte oder 50 Byte an ein Device ausgegeben werden, der Zeitbedarf ist annähernd der Gleiche. Innerhalb Windows benötigt eine einzelne USB-Ausgabe mehrere Millisekunden. Daher ist es effizienter, wenn man für die Ausgabe an die Modelle mit einer Datenstruktur arbeitet, um nicht für jedes Datenbyte einen USB-Zyklus auslösen zu müssen.
Die Ansteuerung der Modelle erfolgt über einen Zyklus: Im ersten Schritt werden die Ausgabedaten aus der in der KeLib gespeicherten Struktur an das Modell gesendet und im zweiten Schritt die Eingänge im Modell gelesen und die Werte in die Struktur geschrieben. Die Funktion "KE_UpdateStructure " führt diese Tätigkeit aus.

Da manche Programmiersprachen Probleme mit der Verarbeitung von Strukturen haben, sind Funktionen vorhanden um die einzelnen Bytes in dieser internen Datenstruktur zu schreiben und zu lesen. Über einen optionalen Parameter kann angegeben werden, ob ein Update mit dem Device erfolgen soll. Hierdurch ist es möglich, zuerst alle Ausgabebytes über mehrere Prozeduraufrufe in die interne Struktur zu schreiben und anschließend die Daten an das Modell zu senden. Dabei findet auch das Einlesen der Eingänge statt. Diese Eingabebytes können dann wiederum über Funktionen einzeln abgefragt werden.

Hinweis:
Die TransferArea kann aber dennoch genutzt werden, sie wird weiterhin auch von der education line unterstützt. Sie bekommen jedoch eine Fehlermeldung, wenn bei laufendem Transfer-Thread versucht wird, mit einer der nachfolgenden Funktionen das Produkt anzusprechen.

4.5.1 DWORD KE_WriteOutputByte (FT_HANDLE hFt, DWORD dwData0 DWORD dwIndex, DWORD dwOption)

USB: ja
COM: nein

Diese Funktion speichert das übergebene Datenbyte (dwData0) in der internen Datenstruktur. Den Aufbau dieser Datenstruktur "KE_EL_IO_STRUCT" findet man im Headerfile "FtLib.h". Die Variable "dwIndex" bestimmt dabei den Speicherplatz (Index) im Array "adwOut". Beispiel: dwIndex = 0 speichert das Datenbyte dwData0 in der Variablen (Speicherstelle) "dwOut0". Die Zuordnung "Datenbytes zu Ausgängen" im Modell wird in der Anleitung des Modells beschrieben.

Über die Variable dwOption kann gesteuert werden, ob das Datenbyte nur in der Struktur gespeichert wird, oder ob anschließend ein USB-Zugriff und somit auch Datenaustausch erfolgen soll. Bei solch einem Datenaustausch werden alle Daten an das Modul gesendet, die Eingänge gelesen und in der Datenstruktur gespeichert.

Aufruf:	FT_HANDLE hFt	Handle des USB-Device
	DWORD dwData0	auszugebendes Datenbyte
	DWORD dwIndex	Index in Array "adwOut" (dwOut0, dwOut1…) der internen KE_EL_IO_STRUCT Datenstruktur
	DWORD dwOption	Bit0: 0 = nur in Struktur speichern 1 = Ausgänge an Modul ausgeben / Eingänge lesen
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler ansonsten ein Fehlercode (s. FtLib.h)
		gängige Fehlercodes:
	FTLIB_ERR_NUMBER_TOO_BIG	falscher Wert in dwIndex

4.5.2 DWORD KE_ReadInputByte (FT_HANDLE hFt, DWORD *pdwData0 DWORD dwIndex, DWORD dwOption)

USB: ja
COM: nein

Diese Funktion schreibt die gewünschten Daten aus der internen Datenstruktur in die Speicherstelle, auf die der Zeiger *pdwData0 verweist. Da die Eingangsdaten in einem Array gespeichert werden, bestimmt dwIndex den Eintrag im Array. Über die Variable dwOption (Bit 0) kann bestimmt werden, ob zuvor noch ein Update-Zyklus mit dem Modul durchgeführt wird (also zuerst die Ausgänge in das Modul schreiben, danach die Eingänge einlesen und in der Struktur speichern). Ferner wird über die Bits 1..3 die Datenquelle (digitale Eingänge, analoge Eingänge) ausgewählt.

Mit dem Parameter dwIndex = "-1" (0xFFFFFFFF) beim Aufruf wird die Versionsnummer der Datenstruktur beim Erstellen der KeLib geliefert, mit dwIndex = "-2" (0xFFFFFFFE) die Sub-Versionsnummer. Da nicht auszuschließen ist, dass es eine Weiterentwicklung in der Datenstruktur für zukünftige Produkte gibt, kann hierüber sichergestellt werden, dass die vorhandene DLL auch zu Ihrem Programm passt.

Aufruf:	FT_HANDLE hFt	Handle des USB-Device
	DWORD *pdwData0	einzulesendes Datenbyte
	DWORD dwIndex	Index in die Eingangs Array´s "adwInp" oder "adwAna" (wird in dwOption festgelegt) der internen KE_EL_IO_STRUCT Datenstruktur Besonderheiten: -1: Version der Datenstruktur bei der Lib-Erstellung -2: Sub-Version der Datenstruktur bei der Lib-Erstellung
	DWORD dwOption	Bit0: 0: nur aus Lib-Struktur lesen 1: Ausgänge an Modul ausgeben / Eingänge lesen Bit1..3: Datenquelle festlegen 000: digitale Eingänge (adwInp) lesen 001: analoge Eingänge (adwAna) lesen
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler ansonsten ein Fehlercode (s. FtLib.h) gängige Fehlercodes:
	FTLIB_ERR_NUMBER_TOO_BIG	falscher Wert in dwIndex

4.5.3 DWORD KE_UpdateStructure (FT_HANDLE hFt, KE_EL_IO_STRUCT *pStruc, DWORD dwOption)

USB: ja
COM: nein

Diese Funktion führt einen Datenaustausch zwischen der internen Lib-Struktur und einer Anwenderstruktur durch, deren Zeiger beim Aufruf übergeben wird. Aus dieser Datenstruktur werden die Ausgabedaten in die interne Lib-Struktur übernommen und die gespeicherten Eingangsdaten von der Lib-Struktur in die Anwenderstruktur übertragen.

Falls beim Aufruf in dwOption Bit0 ("Update") gesetzt ist, werden die Ausgabedaten ins Modul geschrieben und die aktuellen Eingangsdaten in die interne Lib-Struktur und in die Anwenderstruktur übertragen.

Aufruf:	FT_HANDLE hFt	Handle des USB-Device
	KE_EL_IO_STRUCT *pStruc	Zeiger auf die Datenstruktur die mit dem Device abgeglichen wird
	DWORD dwOption	Bit0: Update 0: Daten nur mit Lib-Struktur abgleichen 1: Datenupdate mit Modul durchführen
Return:	Error-Code	
	FTLIB_ERR_SUCCESS	kein Fehler ansonsten ein Fehlercode (s. FtLib.h)

5 Ablauf der USB-Funktionalität für die Online-Kommunikation

Beim Starten der Applikation:
InitFtLib();

Feststellen, welche Geräte angeschlossen sind (Liste erzeugen):
InitFtUsbDeviceList();

Anzahl der gefundenen Geräte ermitteln:
GetNumFtUsbDevice();

Handle auf ein gewünschtes Gerät (Device) holen:
GetFtUsbDeviceHandle(UCHAR ucDevNr) ;
ucDevNr = 0 … GetNumFtUsbDevice()

Feststellen, "welches" Gerät (Typ des Gerätes) angeschlossen ist:
GetFtDeviceTyp(FT_HANDLE hFt);

Öffnen der Datenverbindung zum Device:
OpenFtUsbDevice(FT_HANDLE hFt);

Starten der Kommunikation (Device <-> TransferArea):
StartFtTransferArea(FT_HANDLE hFt, NOTIFICATION_EVENTS* sNEvent);

Abfrage des Zeigers auf die Transfer Area:
GetFtTransferAreaAddress(FT_HANDLE hFt);

.
(eigenes Programm)
.

Stoppen der Kommunikation:
StopFtTransferArea(FT_HANDLE hFt);

Schließen der Datenverbindung zum Device:
CloseFtDevice (FT_HANDLE hFt) oder CloseAllFtUsbDevices()

Beim Beenden der Applikation:
CloseFtLib();

6 Transferarea

Das Abfragen und Steuern der Ausgänge erfolgt durch eine "Transferarea" (Kommunikationsspeicherbereich). Dieser Bereich wird nach dem Starten des Kommunikationsthread (mittels StartFtTransferArea) alle 10ms mit dem Interface abgeglichen. Dabei spielt es keine Rolle, ob das Interface über USB, seriell oder per Funk an den PC angeschlossen ist. Auch spielt es keine Rolle, ob es sich um ein Intelligent-Interface, Robo-Interface oder eine IO-Extension handelt. Über den Devicehandle erkennt die Bibliothek, um welches Produkt es sich handelt und aktualisiert spätestens alle 10ms die vorhandenen Eingänge / Ausgänge des jeweiligen Interfacetyps. Falls mehrere Interface am Rechner angeschlossen sind, kann für jedes der angeschlossenen Produkte ein eigener Thread gestartet werden. Den Aufbau der Transferarea findet man im Anschluß an dieses Kapitel.

Hinweis:
Bei einem Intelligent Interface werden die Daten des Extension-Moduls auf den Speicherplätzen des Extension-Moduls (Erweiterungsmodul) "1" gespeichert.

6.1 Aufbau des Speicherbereichs

Nachdem der Transferthread mittels StartFtTransferArea() gestartet wurde, kann die Basisadresse für den Speicherbereich mit der Funktion GetFtTransferAreaAddress() abgefragt werden.

6.1.1 Speicherlayout des Kommunikationsbereichs

Digitaleingänge des Grundmoduls

	7	6	5	4	3	2	1	0
Base+0x00:	E8	E7	E6	E5	E4	E3	E2	E1

Digitaleingänge der Erweiterungsmodule 1-3

Base+0x01:	E16	E15	E14	E13	E12	E11	E10	E9
Base+0x02:	E24	E23	E22	E21	E20	E19	E18	E17
Base+0x03:	E32	E31	E30	E29	E28	E27	E26	E25

Reserviert (8 Byte)

Base+0x04..0x0B

Reserviert (1 Byte)

Base+0x0C

Reserviert (1 Byte)

Base+0x0D

Sondereingänge vom IR-Sender (Motor 1-3 links/rechts + "1)))" und "2)))")

```
Base+0x0E: | 0 | 0 | 0 | C | T | T | T | T
           C = Code:  0 = Code 1 aktiviert
                      1 = Code 2 aktiviert
             TTTT = Tastennummer 0..11
```

```
Anordnung der Tasten auf Sender:
    1        8
   2     7
  3         10
  4      9
  5        11
  6
```

```
Taste  1 = M3R
Taste  2 = M3L
Taste  3 = Geschwindigkeit M1
Taste  4 = Geschwindigkeit M2
Taste  5 = Geschwindigkeit M3
Taste  6 = Code 2
Taste  7 = M1BW
Taste  8 = M1FW
Taste  9 = M2L
Taste 10 = M2R
Taste 11 = Code 1
```

Reserviert (1 Byte)

```
Base+0x0F
```

Analogeingänge des Grundmoduls (4x 16 Bit, Wertebereich je 0..1023, L:H Format)

```
Base+0x10..0x11: AX (Master Modul)
Base+0x12..0x13: AY (Master Modul)
Base+0x14..0x15: A1 (Master Modul)
Base+0x16..0x17: A2 (Master Modul)
Base+0x18..0x19: AZ (Master Modul, vom SLAVE-Modul-BUS)
Base+0x1A..0x1B: AV (Versorgungsspannung Master Modul)
                     in 10mV Schritten (* 0,01 = Volt)
Base+0x1C..0x1D: D1 (Abstandssensor 1)
Base+0x1E..0x1F: D2 (Abstandssensor 2)
```

Analogeingänge der Extensionmodule 1-3

Base+0x20..0x21: AX (Extension 1 Modul)

Base+0x22..0x23: AX (Extension 2 Modul)

Base+0x24..0x25: AX (Extension 3 Modul)

Reserviert (4 Byte)

Base+0x26..0x27: DS1

Base+0x28..0x29: DS2

Reserviert (2 Byte)

Base+0x2A..0x2B

Reserviert (4 Byte)

Base+0x2C..0x2F

16-Bit Timer 1..6

```
Base+0x30..0x31: Timer 1ms Inkrement

Base+0x32..0x33: Timer 10ms Inkrement

Base+0x34..0x35: Timer 100ms Inkrement

Base+0x36..0x37: Timer 1s Inkrement

Base+0x38..0x39: Timer 10s Inkrement

Base+0x3A..0x3B: Timer 1min Inkrement
```

Reserviert (4 Byte)

```
Base+0x3C..0x3F
```

Ausgänge des Grundmoduls (Polarität)

```
              | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

Base+0x40: |M4B |M4A |M3B |M3A |M2B |M2A |M1B |M1A |
(Nicht vergessen: Bits´s auf Base+0xE1 zu setzen!)
```

Ausgänge der Erweiterungsmodule 1-3 (Polarität)

```
Base+0x41: |M8B |M8A |M7B |M7A |M6B |M6A |M5B |M5A |

Base+0x42: |M12B|M12A|M11B|M11A|M10B|M10A|M9B |M9A |

Base+0x43: |M16B|M16A|M15B|M15A|M14B|M14A|M13B|M13A|
```

Reserviert (4 Byte)

```
Base+0x44..0x47
```

Ausgänge des Grundmoduls (Energiesparmodus aktivieren)

```
              | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

Base+0x48: | 0  | 0  | 0  | 0  | M4 | M3 | M2 | M1 |

     INIT  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |

           1 = Energiesparmodus aktiviert
           bedeutet: wenn beide Ausgänge eines Motors auf "0"
           liegen, wird die Endstufe im Interface "abgeschaltet"
           sodass kein Strom mehr fließt.
```

Ausgänge der Erweiterungsmodule 1-3 (Energiesparmodus deaktivieren)

```
Base+0x49: | 0  | 0  | 0  | 0  | M8 | M7 | M6 | M5 |

Base+0x4A: | 0  | 0  | 0  | 0  | M12| M11| M10| M9 |

Base+0x4B: | 0  | 0  | 0  | 0  | M16| M15| M14| M13|

     INIT  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  |

                1 = Energiesparmodus aktiviert
```

Reserviert (4Byte)

Base+0x4C..0x4F

Ausgänge des Grundmoduls (PWM-Werte, Wertebereich 0…7)

Base+0x50: M1A

Base+0x51: M1B

Base+0x52: M2A

Base+0x53: M2B

Base+0x54: M3A

Base+0x55: M3B

Base+0x56: M4A

Base+0x57: M4B

Ausgänge des IO-Ext1 Modul (PWM-Werte, Wertebereich 0…7)

Base+0x58: M5A

Base+0x59: M5B

Base+0x5A: M6A

Base+0x5B: M6B

Base+0x5C: M7A

Base+0x5D: M7B

Base+0x5E: M8A

Base+0x5F: M8B

Ausgänge des IO-Ext2 Modul (PWM-Werte, Wertebereich 0…7)

Base+0x60: M9A

Base+0x61: M9B

Base+0x62: M10A

Base+0x63: M10B

Base+0x64: M11A

Base+0x65: M11B

Base+0x66: M12A

Base+0x67: M12B

Ausgänge des IO-Ext3 Modul (PWM-Werte, Wertebereich 0…7)

```
Base+0x68: M13A
Base+0x69: M13B
Base+0x6A: M14A
Base+0x6B: M14B
Base+0x6C: M15A
Base+0x6D: M15B
Base+0x6E: M16A
Base+0x6F: M16B
```

Reserviert (32 Bytes)

```
Base+0x70..0x8F
```

Analogeingänge der I/O Extension 1..3 (Update-Time 20ms)
(diese befinden sich auf der rechten 10pol. Stiftleiste, Pin 10, Vorwiderstand von 220..470 Ohm empfohlen)

```
Base+0x90..0x91: A1 (I/O Extension 1 Modul)
Base+0x92..0x93: A1 (I/O Extension 2 Modul)
Base+0x94..0x95: A1 (I/O Extension 3 Modul)

Base+0x96..0x97: AV (Versorgungsspannung I/O Extension 1 Modul)
                     in 10mV Schritten (* 0,01 = Volt)
Base+0x98..0x99: AV (Versorgungsspannung I/O Extension 2 Modul)
                     in 10mV Schritten (* 0,01 = Volt)
Base+0x9A..0x9B: AV (Versorgungsspannung I/O Extension 3 Modul)
                     in 10mV Schritten (* 0,01 = Volt)
```

Reserviert (4 Bytes)

```
Base+0x9C..0x9F
```

Widerstandswerte der Analogeingänge AX / AY

```
Base+0xA0..0xA1: AX (Interface) Resistor-Value (0..5662 Ohm)
Base+0xA2..0xA3: AY (Interface) Resistor-Value (0..5662 Ohm)
Base+0xA4..0xA5: AX (I/O Extension 1 Modul) Resistor-Value (0..5662)
Base+0xA6..0xA7: AX (I/O Extension 2 Modul) Resistor-Value (0..5662)
Base+0xA8..0xA9: AX (I/O Extension 3 Modul) Resistor-Value (0..5662)
```

Reserviert (54 Bytes)

```
Base+0xAA..0xDF
```

Reserviert (1 Byte)

```
Base+0xE0
```

Optionen (RW)

```
Base+0xE1: |    |    |    |    |    |    | UA2| UA1|

     INIT  |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |

               UA1 = 1: Update Ausgänge / PWM alle 10ms
               UA2 = 1: Update Ausgänge / PWM einmalig
Hinweis: Nur wenn UA1 oder UA2 auf "1" sind, werden die
         eingestellten Motorzustände an die Ausgänge geschaltet
         (nur bei Download-Programmen).
```

Nur Windows:

```
Base+0xE2  Kommunikation Windows Thread <-> Windows Applikation
           1 = Thread läuft, 0=Thread steht
           (Byte nur innerhalb der Windows TransferArea verwendet!)
```

Nur Interface: Reserve

```
Base+0xE2  Byte im Interface nicht benutzt
```

Reserviert

```
Base+0xE3..0x0xE5
```

Anzahl angeschlossener I/O Extension Module am Bus (S2..S0 = 0..3)

```
Base+0xE6  |    |    |    |    |    | S2 | S1 | S0 |
```

Reserviert (1 Byte)

```
Base+0xE7:
```

Reserviert (1 Byte)

```
Base+0xE8:
```

Reserviert

```
Base+0XE9..0x0xEF
```

Change Byte Digital-Eingänge

```
Base+0xF0: (1=Eingänge am Master oder IO-Ext1..3 haben sich geändert)
           Windows: InterlockedExchange() Function zum Lesen und
           Schreiben auf 0 verwenden!
```

Change Byte Analog-Eingänge

Master: AX, AY, A1, A2, AV, AZ, D1, D2 oder I/O Extension 1..3 AX, AY, AV haben sich geändert

```
Base+0xF1: (1=Analogeingänge am Master oder IO-Ext1..3 haben sich
           geändert) Windows: InterlockedExchange() Function zum
           Lesen und Schreiben auf 0 verwenden!
```

Change Byte Ir-Eingänge

```
Base+0xF2: (1=Änderung am IR-Sensoreingang)
       Windows: InterlockedExchange() Function zum
       Lesen und Schreiben auf 0 verwenden!
```

Reserve

```
Base+0XF3..0x0xFF
```

Nachfolgend sind die (Digital-) Eingänge nochmals pro Eingang in einer 16-Bit Variablen abgelegt. Ist der Eingang mit "+" verbunden, wird dies als "1" gewertet. Ein offener oder mit "Masse" verbundener Eingang wird als "0" gewertet.

```
Base+0x100..0x101  Eingang  1 (Master-Modul)
Base+0x102..0x103  Eingang  2 (Master-Modul)
Base+0x104..0x105  Eingang  3 (Master-Modul)
Base+0x106..0x107  Eingang  4 (Master-Modul)
Base+0x108..0x109  Eingang  5 (Master-Modul)
Base+0x10A..0x10B  Eingang  6 (Master-Modul)
Base+0x10C..0x10D  Eingang  7 (Master-Modul)
Base+0x10E..0x10F  Eingang  8 (Master-Modul)
Base+0x110..0x111  Eingang  9 (Slave1-Modul)
Base+0x112..0x113  Eingang 10 (Slave1-Modul)
Base+0x114..0x115  Eingang 11 (Slave1-Modul)
Base+0x116..0x117  Eingang 12 (Slave1-Modul)
Base+0x118..0x119  Eingang 13 (Slave1-Modul)
Base+0x11A..0x11B  Eingang 14 (Slave1-Modul)
Base+0x11C..0x11D  Eingang 15 (Slave1-Modul)
Base+0x11E..0x11F  Eingang 16 (Slave1-Modul)
Base+0x120..0x121  Eingang 17 (Slave2-Modul)
Base+0x122..0x123  Eingang 18 (Slave2-Modul)
Base+0x124..0x125  Eingang 19 (Slave2-Modul)
Base+0x126..0x127  Eingang 20 (Slave2-Modul)
Base+0x128..0x129  Eingang 21 (Slave2-Modul)
Base+0x12A..0x12B  Eingang 22 (Slave2-Modul)
Base+0x12C..0x12D  Eingang 23 (Slave2-Modul)
Base+0x12E..0x12F  Eingang 24 (Slave2-Modul)
Base+0x130..0x131  Eingang 25 (Slave3-Modul)
Base+0x132..0x133  Eingang 26 (Slave3-Modul)
Base+0x134..0x135  Eingang 27 (Slave3-Modul)
Base+0x136..0x137  Eingang 28 (Slave3-Modul)
Base+0x138..0x139  Eingang 29 (Slave3-Modul)
Base+0x13A..0x13B  Eingang 30 (Slave3-Modul)
Base+0x13C..0x13D  Eingang 31 (Slave3-Modul)
Base+0x13E..0x13F  Eingang 32 (Slave3-Modul)
Base+0x140..0x141  Abstandssensor D1 (Master-Modul)
Base+0x142..0x143  Abstandssensor D2 (Master-Modul)
```

Reserviert (12 Bytes)

```
Base+0x140..0x14F
```

Die IR-Tasten werden nachfolgend als einzelne Eingänge in einer 16-Bit Variablen pro Taste dargestellt (1=Taste gedrückt). Eine gedrückte Taste wird mit einer "1" sowohl im "undecodierten" Bereich, wie auch im "decodierten" Bereich angezeigt.

```
Base+0x150..0x151  IR Taste  1 (M3R)
Base+0x152..0x153  IR Taste  2 (M3L)
Base+0x154..0x155  IR Taste  3 (M1)
Base+0x156..0x157  IR Taste  4 (M2)
Base+0x158..0x159  IR Taste  5 (M3)
Base+0x15A..0x15B  IR Taste  6 (Code2)
Base+0x15C..0x15D  IR Taste  7 (M1BW)
Base+0x15E..0x15F  IR Taste  8 (M1FW)
Base+0x160..0x161  IR Taste  9 (M2LE)
Base+0x162..0x163  IR Taste 10 (M2RI)
Base+0x164..0x165  IR Taste 11 (Code1)
```

Reserviert (10 Bytes)

```
Base+0x166..0x16F
```

```
Base+0x170..0x171  IR Taste  1 (M3R)    Code1
Base+0x172..0x173  IR Taste  2 (M3L)    Code1
Base+0x174..0x175  IR Taste  3 (M1)     Code1
Base+0x176..0x177  IR Taste  4 (M2)     Code1
Base+0x178..0x179  IR Taste  5 (M3)     Code1
Base+0x17A..0x17B  reserviert
Base+0x17C..0x17D  IR Taste  7 (M1BW)   Code1
Base+0x17E..0x17F  IR Taste  8 (M1FW)   Code1
Base+0x180..0x181  IR Taste  9 (M2LE)   Code1
Base+0x182..0x183  IR Taste 10 (M2RI)   Code1
Base+0x184..0x185  reserviert
```

Reserviert (10 Bytes)

```
Base+0x166..0x18F
```

```
Base+0x190..0x191  IR Taste  1 (M3R)    Code2
Base+0x192..0x193  IR Taste  2 (M3L)    Code2
Base+0x194..0x195  IR Taste  3 (M1)     Code2
Base+0x196..0x197  IR Taste  4 (M2)     Code2
Base+0x198..0x199  IR Taste  5 (M3)     Code2
Base+0x19A..0x19B  reserviert
Base+0x19C..0x19D  IR Taste  7 (M1BW)   Code2
Base+0x19E..0x19F  IR Taste  8 (M1FW)   Code2
Base+0x1A0..0x1A1  IR Taste  9 (M2LE)   Code2
Base+0x1A2..0x1A3  IR Taste 10 (M2RI)   Code2
Base+0x1A4..0x1A5  reserviert
```

Reserviert (10 Bytes)

```
Base+0x1A6..0x1AF
```

RF-Status (nur RF-Data-Link USB-Modul)

```
Base+0x1B0..0x1B1:    0 = RF-Verbindung in Ordnung
                      1 = (0x1B4..0x1B5) > 25, schlechte Verbindung
Base+0x1B2..0x1B3:    Empfangsqualität, bei <40 schlechte Verbindung
                      (8 Bit-Wert)
Base+0x1B4..0x1B5:    Fehlerzähler bei RF-Online Betrieb, wird bei
                      fehlerhafter Verbindung erhöht und auf "0"
                      gesetzt wenn Verbindung in Ordnung.
```

Reserviert

```
Base+0x1B6..0x1FF
```

6.1.2 Digitaleingänge E1-E32

Die Bits für die Digitaleingänge werden bei einem offen Eingang auf "0", bei einem mit „+“ verbundenen Eingang auf "1" gesetzt. Nicht vorhandene Eingänge (fehlende Erweiterungsmodule) werden auf "0" gesetzt. Zusätzlich sind alle 32 Eingänge nochmals ab Base+0x100 in einer 16-Bit Variablen pro Eingang abgelegt ("1"=Eingang betätigt).

6.1.3 Sondereingänge

Sondereingänge sind die 11 Tasten der IR-Fernbedienung. Die Nummer der am IR-Sender gedrückten Taste, sowie die Information ob Code „1“ oder Code „2“ aktiviert wurde, wird in die Speicherstelle Base+0x0E abgespeichert. Zusätzlich werden alle Tasten nochmals in 16 Bit-Variablen (wie die Digitaleingänge) abgespeichert.

6.1.4 Analogeingänge

Die Analogeingänge werden als 16 Bit-Werte mit einem Wertebereich von 0…1023 abgelegt.

6.1.5 16 Bit Timer

Die 6 16 Bit-Timer mit Inkrementen von 1ms, 10ms, 100ms, 1s, 10s und 60s werden für bestimmte Timeout-Variablen verwendet. Zwischen den einzelnen Timerwerten besteht keine feste Beziehung, d.h. der 10ms Wert ist z.B. nicht 10x der 1ms Wert.

6.1.6 Ausgänge

Die Ausgänge werden über ein Polaritätsbit, ein Energiesparbit und ein PWM-Wert- Byte gesteuert. Der PWM-Wert und das Polaritätsbit werden pro Einzelausgang angeben. Das Energiesparbit wird pro Ausgangspaar angegeben. Ist das Polaritätsbit "0", wird der Ausgang auf Masse gesetzt, ist das Polaritätsbit "1" wird der Ausgang auf Versorgung (9V) gesetzt. Ist das Energiesparbit "1" wird ein Ausgangspaar nach einer Verzögerung (1 sek.) hochohmig geschaltet, wenn beide zugehörigen Polaritätsbits auf "0" gesetzt sind. Ist das Energiesparbit auf "0" gesetzt, wird das zugehörige Ausgangspaar nicht hochohmig geschaltet. Über das PWM-Byte, das einen Wertebereich von 0…7 hat, wird die Pulsbreite des Ausgangs in 8 Stufen eingestellt (z.B. in 12.5% Schritten zwischen 12.5% und 100%).

Hinweis für Downloadprogramme:
Die Ausgangseinstellungen werden von der Firmware alle 10ms in einen eigenen Datenbereich kopiert, wenn das Bit "UpdateAusgänge" (UA1) auf BASE+0xE1 gesetzt ist. Wird anstelle UA1, UA2 gesetzt, dann werden die Ausgänge nur einmalig beim nächsten 10ms Interrupt geschrieben. Nachdem die Ausgänge gesetzt sind, wird das UA2 Bit gelöscht.

Nach der Initialisierung des Interface (einschalten) sind alle Energiesparbits auf "1" gesetzt. Standardmäßig ist diese Funktionalität aktiviert.

6.1.7 Betriebsmodus, installierte Erweiterungen

Ab Base+0x0E finden sich Informationen über den Betriebszustand, sowie über den Typ der installierten Module (Anzahl I/O Extensions, Funkmodul, usw.) wieder.

7 Revision

Version 0.56: - Komplette Überarbeitung
- Neu: SendFtMessage()
ClearFtMessagePuffer()
StartDtTransferAreaWithCommunication()
SetFtDeviceCommMode()

Version 0.58: - Überarbeitung von InitFtUsbDeviceList()

Version 0.60: - Umbenennung GetAnzFtUsbDevice() in GetNumFtUsbDevice()
- Umbenennung ClearFtMessagePuffer() in ClearFtMessageBuffer()

Version 1.61a: - Umbenennung der FtLib von 0.60 in nun 1.61a
- Unterstützung für Interface-Firmware 01.66.00.03

Version 1.70a: - Überarbeitung SetFtDistanceSensorMode()
- Unterstützung für Interface-Firmware 01.75.00.04

Version 1.74: - Unterstützung "education line" (neue Funktionen)