FAULHABER

Bibliothekshandbuch

Kommunikations-API

MomanLib

Impressum

Version:
2. Auflage, 8.10.2022

Softwarestand:
V2.0, entsprechend "FAULHABER Motion Manager 6.9"

Dieses Dokument wurde mit Sorgfalt erstellt.
Die Dr. Fritz Faulhaber GmbH & Co. KG übernimmt jedoch für eventuelle Irrtümer in diesem Dokument und deren Folgen keine Haftung. Ebenso wird keine Haftung für direkte Schäden oder Folgeschäden übernommen, die sich aus einem unsachgemäßen Gebrauch der Geräte ergeben.

Bei der Anwendung der Geräte sind die einschlägigen Vorschriften bezüglich Sicherheitstechnik und Funkentstörung sowie die Vorgaben dieses Dokuments zu beachten.

Änderungen vorbehalten.

Die jeweils aktuelle Version dieses Dokuments finden Sie auf der Internetseite von FAULHABER:
www.faulhaber.com

Inhalt

Inhalt

1 Zu diesem Dokument

1.1 Gültigkeit dieses Dokuments

Dieses Dokument beschreibt die Anwendungsprogrammierschnittstelle **MomanLib** zur Programmierung einer Steuerungssoftware für FAULHABER Motion Controller unter Microsoft Windows.

Dieses Dokument richtet sich an ausgebildete Programmierer und Informatiker.

1.2 Mitgeltende Dokumente

Für die Verwendung der API sind zusätzliche Informationen aus folgenden Handbüchern hilfreich:

Handbuch	Beschreibung
Motion Manager 6	Bedienungsanleitung zur FAULHABER Motion Manager PC Software
Kommunikationshandbuch	Beschreibung der Kommunikation mit dem Antrieb
Antriebsfunktionen	Beschreibung der Betriebsarten und Funktionen des Antriebs

Diese Handbücher können im PDF-Format von der Internetseite www.faulhaber.com/manuals

heruntergeladen werden.

1.3 Symbole und Kennzeichnungen

Hinweise zum Verständnis oder zum Optimieren der Arbeitsabläufe

✓ Voraussetzung zu einer Handlungsaufforderung

1. Erster Schritt einer Handlungsaufforderung
   ↳ Resultat eines Schritts
2. Zweiter Schritt einer Handlungsaufforderung

↳ Resultat einer Handlung

▸ Einschrittige Handlungsaufforderung

1.4 Abkürzungsverzeichnis

Abkürzung	Bedeutung
API	Anwendungsprogrammierschnittstelle
CAN	Controller Area Network
CiA	CAN in Automation e.V.
COM	Serielle RS232-Schnittstelle
DLL	Dynamic Link Library
MC	Motion Controller
NMT	CANopen Netzwerkmanagement
OD	Objektverzeichnis
PC	Personal Computer
USB	Universal Serial Bus

1.5 Rechtliche Hinweise

Gewerbliche Schutzrechte
Mit der Veröffentlichung der Motion Manager Library werden weder ausdrücklich noch konkludent Rechte an gewerblichen Schutzrechten, die mittelbar oder unmittelbar den beschriebenen Anwendungen und Funktionen der Motion Manager Library zugrunde liegen, übertragen noch Nutzungsrechte daran eingeräumt.

Kein Vertragsbestandteil; Unverbindlichkeit der Motion Manager Library
Die Motion Manager Library ist nicht Vertragsbestandteil von Verträgen, die die Dr. Fritz Faulhaber GmbH & Co. KG abschließt, soweit sich aus solchen Verträgen nicht etwas anderes ergibt. Die Motion Manager Library beschreibt unverbindlich ein mögliches Anwendungsbeispiel. Die Dr. Fritz Faulhaber GmbH & Co. KG übernimmt insbesondere keine Garantie dafür und steht insbesondere nicht dafür ein, dass die in der Motion Manager Library illustrierten Abläufe und Funktionen stets wie beschrieben aus- und durchgeführt werden können und dass die in der Motion Manager Library beschriebenen Abläufe und Funktionen in anderen Zusammenhängen und Umgebungen ohne zusätzliche Tests oder Modifikationen mit demselben Ergebnis umgesetzt werden können.

Keine Haftung
Die Dr. Fritz Faulhaber GmbH & Co. KG weist darauf hin, dass aufgrund der Unverbindlichkeit der Motion Manager Library keine Haftung für Schäden übernommen wird, die auf die Motion Manager Library zurückgehen.

Änderungen der Motion Manager Library
Änderungen der Motion Manager Library sind vorbehalten. Die jeweils aktuelle Version dieser Motion Manager Library erhalten Sie von Dr. Fritz Faulhaber GmbH & Co. KG unter der Telefonnummer +49 7031 638 688 oder per Mail von mcsupport@faulhaber.de.

2 Beschreibung

Die API **MomanLib** stellt eine einheitliche Funktions-Schnittstelle zur Verfügung, um über verschiedene Schnittstellen (USB, RS232, CAN) mit FAULHABER Motion Controllern zu kommunizieren.

Die API kann zur Programmierung einer 32-Bit Steuerungs-Software unter Microsoft Windows mit einer selbst gewählten Programmiersprache (z. B. C++, C#, Delphi, LabVIEW) verwendet werden.

Die API-Funktionen sind in C++ geschrieben. Zur Benutzung mit anderen Programmiersprachen müssen die API-Funktionen über passende Wrapper-Funktionen eingebunden werden. Für gängige Programmiersprachen sind Beispiele gegeben.

2.1 Architektur

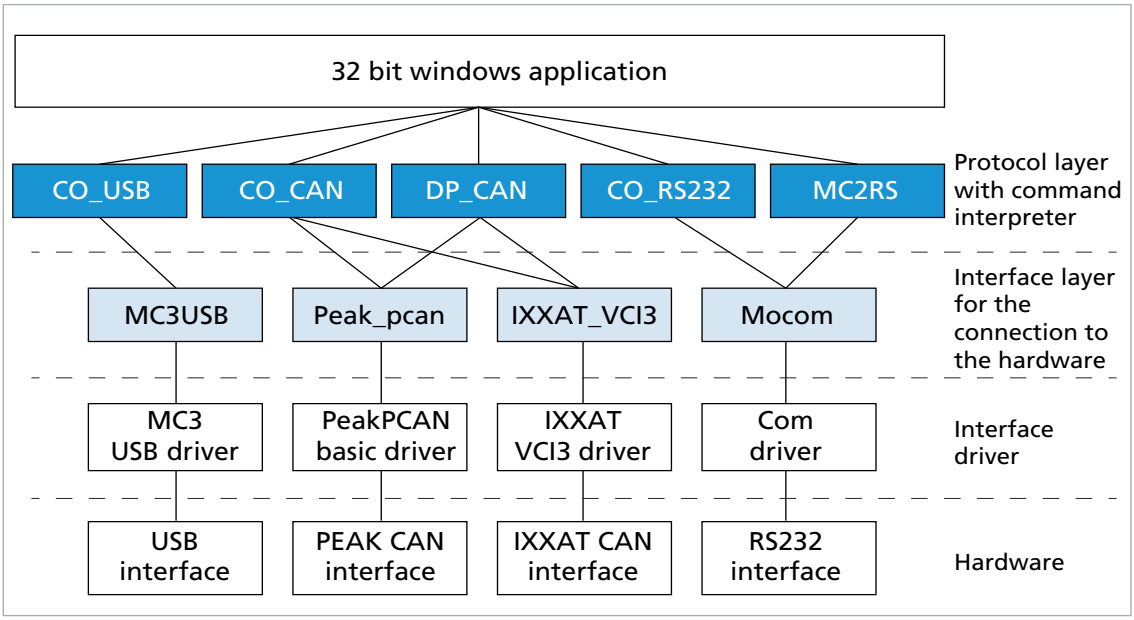


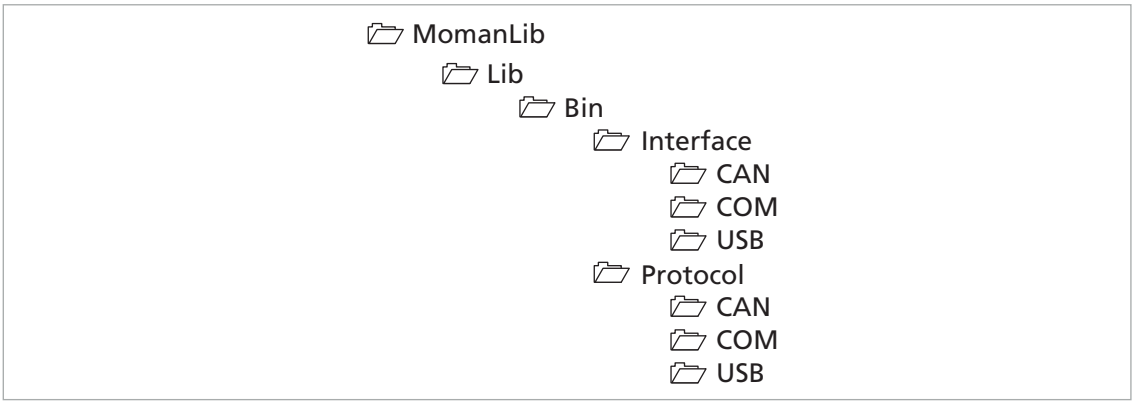
Abb. 1: Architektur der API

Die zweischichtige Plug-in-Architektur erlaubt die Unterstützung unterschiedlicher Kommunikations-Protokolle und Interface-Karten für die RS232-, USB- und CAN-Schnittstelle.

Jede Schicht wird durch eine DLL-Datei, eine Protokoll-DLL und eine Interface-DLL, repräsentiert. Die Kommunikation zwischen den Schichten erfolgt über eine definierte Funktionsschnittstelle.

2.2 Dateien

Die für die Verwendung der API notwendigen Dateien sind in diesem Paket im Unterverzeichnis \MomanLib\Lib\Bin enthalten. Das Bin-Verzeichnis enthält die Unterverzeichnisse \Protocol und \Interface, die jeweils Unterverzeichnisse für die unterstützten Schnittstellen mit den jeweiligen DLL-Dateien enthalten:



Für die Verwendung der API in eigenen Programmen wird empfohlen, die benötigten Dateien in das eigene Projektverzeichnis zu kopieren und von dort zu verwenden.

Immer zwei DLL-Dateien werden benötigt:

- Protokoll-DLL
- Interface-DLL

Abhängig von der Art des Protokolls und der Schnittstelle stehen unterschiedliche DLL-Dateien zur Verfügung:

Tab. 1: Protokoll-DLL

Schnittstelle	Protokoll-DLL	Bedeutung
RS232	MC2RS	Serielles Protokoll für Motion Controller der Familie MC V2.x
	CO_RS232	CO-Protokoll über RS232 für Motion Controller der Familie MC V3.x
CAN	CO_CAN	Standard CANopen-Protokoll über CAN
USB	CO_USB	CO-Protokoll über USB für Motion Controller der Familie MC V3.x

Tab. 2: Interface-DLL

Schnittstelle	Interface-DLL	Funktion
RS232	Mocom.dll	Verbindung zu Standard Serial COM-Port
CAN	Ixxat_vci3.dll	Verbindung zu HMS-IXXAT VCI3- / VCI4-Treiber
	Peak_pcan.dll	Verbindung zu PEAK PCAN-Treiber
	Ems_cpc.dll	Verbindung zu EMS CPC-Treiber
	Esd_ntcan.dll	Verbindung zu ESD NTCAN-Treiber
	…	Bei Bedarf weitere CAN-Interface-DLLs
USB	MC3Usb.dll	Verbindung zu FAULHABER MC V3.x USB-Treiber

USB
Der Treiber für den Zugriff auf den FAULHABER Motion Controller der Familie MC V3.x wird mit dem Motion Manager 6 installiert.

RS232
Bei Verwendung einer im PC eingebauten seriellen Schnittstelle ist kein weiterer Treiber erforderlich. Wenn ein USB-to-Serial-Adapter verwendet wird, muss in der Regel der mitgelieferte Treiber installiert werden. Der Adapter erscheint dann als virtueller COM-Port, dessen Port-Nummer aus dem Windows-Gerätemanager entnommen werden kann (z. B. COM5).

CAN
Der Treiber der verwendeten CAN-Interface-Karte, passend zur ausgewählten Interface-DLL, muss separat installiert sein. Unterstützt werden aktuell CAN-Interface-Karten der in Tab. 2 angegebenen Hersteller. CAN-Interface-Karten anderer Hersteller können verwendet werden, wenn eine Interface-DLL hierfür existiert. Dies kann bei FAULHABER angefragt werden.

Für einige Interfaces wird eine zusätzliche DLL im Anwendungsverzeichnis benötigt, wie z. B. für PEAK die PCANBasic.dll. Die zusätzlichen DLL-Dateien finden Sie im Ordner \Examples\Win32\Common.

3 Einbindung in die Anwendung

Die Protokoll-DLL kann statisch oder dynamisch in Windows-Anwendungen, die mit Win32-Entwicklungswerkzeugen (z. B. C/C++, Delphi, Visual Basic, LabVIEW) entwickelt werden, eingebunden werden.

Die gewünschten DLL-Dateien (Protokoll-DLL und Interface-DLL) müssen in das Projekt-Verzeichnis oder ein passendes Unterverzeichnis kopiert werden.

Zusätzlich müssen die zu verwendenden Funktionen, die in der C-Header-Datei **Momanprot.h** definiert sind, der Anwendung bekannt gemacht werden. Für die Verwendung dieser Funktionen werden Definitionen aus der Datei **Momancmd.h** benötigt, die der Anwendung ebenfalls bekannt gemacht werden müssen. Die C-Header-Dateien befinden sich im Verzeichnis \MomanLib\Lib\Include.

Im Programm muss zuerst über die Funktion **`mmProtInitInterface()`** die gewünschte Interface-DLL initialisiert werden. Danach können die Schnittstelle geöffnet, Kommandos gesendet und Antworten eingelesen werden. Am Ende müssen Schnittstelle und Interface wieder geschlossen werden.

3.1 Typische Aufruffolge

3.1.1 Synchrone Kommunikation

1. Initialisierung

   Verbindung zu einem Motion Controller der Familie MC V3.x über USB:

   ```
   mmProtInitInterface("MC3Usb.dll", NULL, NULL);
   mmProtOpenCom(1, 0, 0);
   ```

2. Datenaustausch

   Parameter "Device Name" von Knoten 1 auslesen:

   ```
   const char* answData = NULL;
   mmProtGetStrObj(1, 0x1008, 0x00, &answData);
   ```

3. Schnittstelle und Interface schließen:

   ```
   mmProtCloseCom();
   mmProtCloseInterface();
   ```

3.1.2 Asynchrone Kommunikation

1. Initialisierung

   Verbindung zu einem Motion Controller der Familie MC V3.x über eine HMS-IXXAT-CAN-Karte mit 250 kBit/s:

```
mmProtInitInterface("Ixxat_vci3.dll", &CBDataReceived, NULL);
mmProtOpenCom(1, 0, 250000);
hReceiveEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
```

2. Datenaustausch

   Knoten 1 in Zustand "Operation Enabled" versetzen und Daten asynchron empfangen:

```
mmProtSendCommand(1, 0x0000, eMomancmd_start, 0, 0);
mmProtSendCommand(1, 0x0000, eMomancmd_shutdown, 0, 0);
mmProtSendCommand(1, 0x0000, eMomancmd_switchon, 0, 0);
mmProtSendCommand(1, 0x0000, eMomancmd_EnOp, 0, 0);
// Callback function for the signalling of data reception:
void CBDataReceived(void)
{
   SetEvent(hReceiveEvent);
}
```

3. Daten einlesen (hier in eigenem Thread mit Event-Signalisierung):

```
if (WaitForSingleObject(hReceiveEvent, INFINITE) == WAIT_OBJECT_0)
{
   const char* answData = NULL;
   const char* cmdString = NULL;
   const char* receiveTelegram = NULL;
   int nodeNr;
   mmProtReadAnswer(&answData, nodeNr, &cmdString, &receiveTelegram);
}
```

4. Schnittstelle und Interface schließen:

```
mmProtCloseCom();
mmProtCloseInterface();
CloseHandle(hReceiveEvent);
```

3.2 C/C++

Für die statische Einbindung muss eine passende Import-Library der Protokoll-DLL (*.lib) zum Projekt hinzugefügt werden. Da es unterschiedliche Formate von Import-Libraries abhängig von der eingesetzten Entwicklungsumgebung gibt, wird diese Methode in den Beispielen nicht verwendet. Für manche Entwicklungsumgebungen gibt es Tools zur Erzeugung einer Import-Library aus einer DLL (siehe Dokumentation der eingesetzten Entwicklungsumgebung).

Für die dynamische Einbindung muss zuerst über die Win32-Funktion **`LoadLibrary()`** die gewünschte Protokoll-DLL geladen und dann jede zu verwendende Funktion über **`GetProcAddress()`** auf einen äquivalenten Funktionsnamen gelegt werden. Nach Ende der Benutzung muss die DLL wieder über **`FreeLibrary()`** aus dem Speicher entfernt werden.

Für den Zugriff auf die DLL-Funktionen müssen die Header-Dateien **Momanprot.h** und **Momancmd.h** in das Projektverzeichnis kopiert und per `#include` in die Quelltextdatei aufgenommen werden.

Beispiel (\MomanLib\Examples\Source\C++)

- \Common\MomanLibSample

  Zeigt die dynamische Einbindung der Library in ein Standard C++-Programm und die Verwendung einzelner Funktionen

- \C++Builder\DemoVCL

  C++-Builder-Projekt für Embarcadero RAD Studio 10 mit VCL-Bedienoberfläche zur Benutzung von MomanLibSample

- \Visual C++\DemoVCPP

  Visual C++-Projekt für Microsoft Visual Studio 2012 mit "Windows Forms"-Bedienoberfläche zur Benutzung von MomanLibSample

3.3 Delphi

Für die statische Einbindung müssen in der Quelltextdatei die zu verwendenden DLL-Funktionen als **extern** mit Aufrufkonvention `stdcall` deklariert werden. Diese bevorzugte Methode wird auch im Beispiel verwendet.

Die Library kann auch dynamisch über die Win32-Funktion **`LoadLibrary()`** geladen werden. Die Einbindung der Funktionen erfolgt dann wie bei C++ über **`GetProcAddress()`**.

Für die Verwendung der DLL-Funktionen werden einige Definitionen aus der Datei **Momancmd.h** benötigt, die als `type` in der Quelltextdatei angegeben werden müssen. Bei der Verwendung von Callback-Funktionen (asynchrone Kommunikation) ist zu beachten, dass diese mit der Aufrufkonvention CDECL deklariert werden müssen.

Beispiel (\MomanLib\Examples\Source\Delphi)

- \MomanLibSample

  Zeigt die statische Einbindung und Verwendung einzelner Funktionen der Library in ein Delphi-Programm

- \Demo

  Delphi-Projekt für Embarcadero RAD Studio 10 mit VCL-Bedienoberfläche zur Benutzung von MomanLibSample

3.4 C#

Für die Einbindung muss eine Wrapper-Klasse erstellt werden, mit den `DllImport`-Verweisen auf die benötigten Funktionen der angegebenen DLL. Als Aufrufkonvention muss `CallingConvention=CallingConvention.StdCall` angegeben sein.

Für die Verwendung der DLL-Funktionen werden einige Definitionen aus der Datei **Momancmd.h** benötigt, die in der Quelltextdatei angegeben werden müssen. Bei der Verwendung von Callback-Funktionen (asynchrone Kommunikation) ist zu beachten, dass diese mit der Aufrufkonvention CDECL deklariert werden müssen.

Beispiel (\MomanLib\Examples\Source\C#)

- \MomanLibSample

  Zeigt die Einbindung und Verwendung einzelner Funktionen der Library in ein C#-Programm

- \DemoCSharp

  C#-Projekt für Microsoft Visual Studio 2012 mit "Windows Forms"-Bedienoberfläche zur Benutzung von MomanLibSample

3.5 LabVIEW

Für die Einbindung muss im "Knoten zum Aufruf externer Bibliotheken"/"Call Library Function Node" der Pfad der DLL-Datei angegeben und die zu verwendende DLL-Funktion ausgewählt und konfiguriert werden. Als Aufrufkonvention muss `stdcall` angegeben werden.

Für die Verwendung dieser Funktionen muss "Pfad im Blockdiagramm angeben"/"Specify path on diagram" ausgewählt sein. Außerdem müssen noch einige Definitionen aus der Datei **Momancmd.h** angegeben werden. Bei der Verwendung von Callback-Funktionen (asynchrone Kommunikation) ist zu beachten, dass diese mit der Aufrufkonvention CDECL deklariert werden müssen.

Bei der Verwendung der automatischen Importfunktion muss darauf geachtet werden, dass die Import-Deklarationen korrekt importiert worden sind. Es bietet sich an, die entsprechenden Rückgabewerte sicher abzufangen.

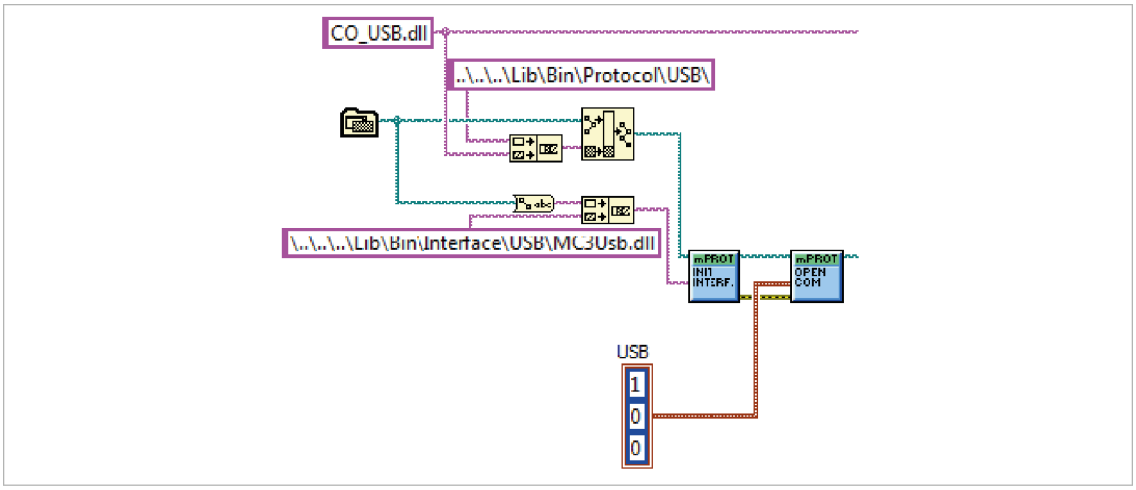


Abb. 2: Initialisierung mit Wrapper-Baustein

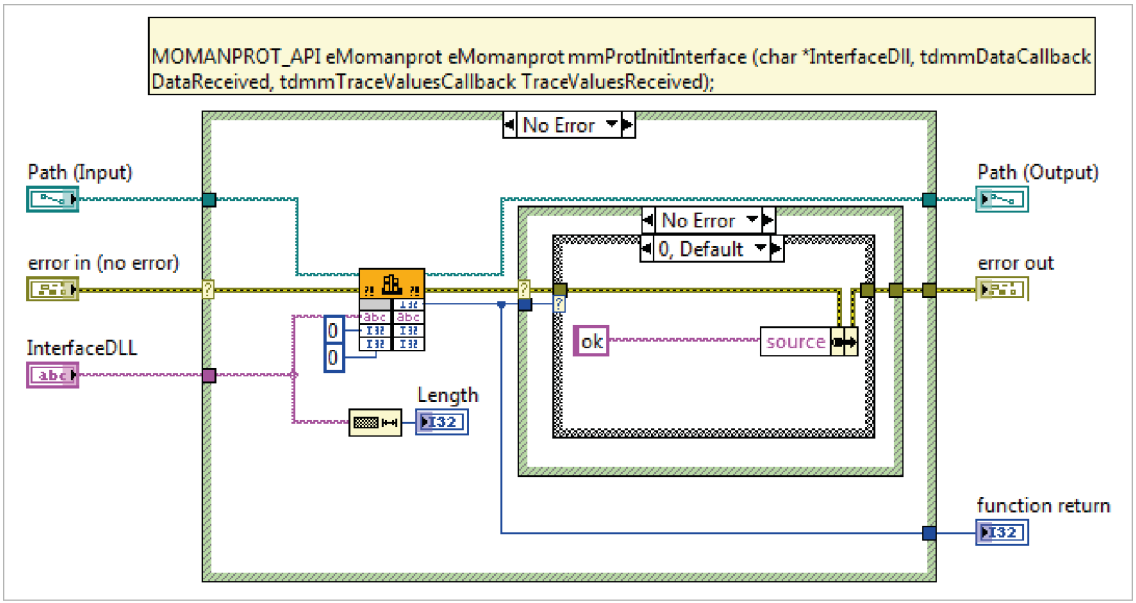


Abb. 3: Implementation eines Wrapper-Bausteins

Die Asynchrone Kommunikation (siehe Kap. 3.1.2, S. 11) ist im Beispiel nur vorbereitet. Für die Verwendung mit LabView ist eine Erweiterung der API-Schnittstelle notwendig.

Beispiel (\MomanLib\Examples\Source\Labview)

- Projekt für LabView 2015 mit Bedienoberfläche und Hauptschleife
- \Typedefinition

  Definitionen verwendeter API-Parameter
- \SubVIs
  - \Object-convert.vi, \Command_format_WRITE.vi

    Vorformatierung der API-Parameter
  - \mmProt___.vi

    Wrapper für jeden "Call Library Function Node"-Funktionsblock
  - \Async___.vi, \.NET_Init.vi

    Vorbereitungen für die Verwendung der Asynchronfunktion

4 Beispiele

Für die in Kap. 3, S. 10 aufgeführten Programmiersprachen sind einfache Beispiele verfügbar.

Der Quellcode für die einzelnen Programmiersprachen liegt im Unterordner \Examples\Source. Die zu Grunde liegenden C-Header-Dateien liegen in \Lib\Include. Für einige Programmiersprachen gibt es eine Datei mit dem Namen *MomanLibSample*, die die Kommunikations-DLLs lädt und den Zugriff auf die API-Funktionen in entsprechenden Wrapper-Funktionen kapselt. Zusätzlich sind hier die benötigten Definitionen aus der Datei **Momancmd.h** für andere als C/C++-Programmiersprachen implementiert.

Im Unterordner \Examples\Win32 liegen die mit den jeweiligen Programmiersprachen erzeugten ausführbaren EXE-Dateien, die die Kommunikations-DLLs aus dem Ordner \Lib\Bin verwenden.

Die Beispiele sind so aufgebaut, dass sie leicht zu verstehen sind. Auf ausreichende Fehlerbehandlung und weitere Modularisierung wurde im Sinne einer leichteren Lesbarkeit des Codes häufig verzichtet. Die Beispiele sollten vor der Benutzung an die eigene Softwarestruktur angepasst werden.

Die Beispiele zeigen folgende Aspekte:

- Synchroner Zugriff auf Objekte im Objektverzeichnis
- Bedienung der Device-Control-Zustandsmaschine mit asynchronem Datenempfang
- Ausführung einer relativen Positionierung

Alle Beispiele verwenden USB als Kommunikationsschnittstelle zu einem Motion Controller der Familie V3.x. Für die Ausführung der Beispiele muss eine USB-Verbindung zu einem Motion Controller der Familie V3.x hergestellt sein. Wenn die Verbindung über eine andere Schnittstelle hergestellt werden soll, müssen die dafür notwendigen Protokoll- und Interface-DLLs den jeweiligen Konstanten im Quellcode zugewiesen werden.

Vor der Ausführung eines Beispielprogramms muss der Motion Controller über den Motion Manager an den angeschlossenen Motor angepasst werden.

5 API-Dokumentation

Die Protokoll-DLL beinhaltet die Schnittstelle zur Anwendung. Auch die Interface-DLL wird hierüber initialisiert.

5.1 Allgemein

In der Datei **Momancmd.h** sind mehrere Enumeratoren und Strukturen definiert, die als Rückgabe- oder Übergabeparameter von einigen Funktionen verwendet werden.

In der Funktionsbeschreibung ist angegeben, welche Enumeratoren und Strukturen die jeweiligen Funktionen verwenden. Diese Datei kann direkt in C/C++-Programme per `#include` eingebunden werden. Für andere Programmiersprachen müssen die verwendeten Enumeratoren und Strukturen entsprechend bekannt gemacht werden.

In der Datei **Momanprot.h** ist die API-Definition für die Programmiersprache C/C++ hinterlegt.

Einige Funktionen erwarten einen Doppelzeiger `const char**` als Übergabeparameter. Dieser Zeiger ermöglicht das Auslesen von Strings, deren Speicher in der Protokollschicht verwaltet wird. Für die Weiterverarbeitung dieser Strings muss der hiermit referenzierte Speicherbereich direkt nach Aufruf der Funktion in eine passende String-Variable umkopiert werden.

Beispiele

C++: Funktionsparameter definiert als (`const char** data`)

Funktionsaufruf:

```
const char* data = NULL;
DoSomething(&data);
if (data != NULL) {
    std::string s = data;
}
```

C#: Funktionsparameter definiert als (`out IntPtr data`)

Funktionsaufruf:

```
IntPtr data = IntPtr.Zero;
DoSomething(out data);
if (data != IntPtr.Zero) {
    string s = Marshal.PtrToStringAnsi(data)
}
```

Delphi: Funktionsparameter definiert als (*data:* PAnsiChar)

Funktionsaufruf:

```
s: String;
data: PAnsiChar;
data := nil;
DoSomething(@data);
if data <> nil then begin
    s = String(data)
end;
```

LabVIEW: Funktionsparameter definiert als (uintptr_t *data)

Funktionsaufruf („Knoten zum Aufruf externer Bibliotheken“/„Call Library Function Node“):

GetValueByPointer VI

```
Type: Numeric
Datatype: unsigned Pointer-sized Integer
Pass: Pointer to Value
```

Die Motion Manager API verwendet generell 8-Bit ANSI-Strings. Für die Verwendung von Unicode-Strings müssen entsprechende Umwandlungen durchgeführt werden.

5.2 Initialisierung

Die Funktionen zur Initialisierung sind für einen Verbindungsaufbau zwischen der PC-Software und dem Motion Controller notwendig.

5.2.1 InitInterface

```
eMomanprot mmProtInitInterface ( char*                   InterfaceDll,
                                 tdmmDataCallback        DataReceived,
                                 tdmmTraceValuesCallback TraceValuesReceived
                               )
```

Initialisierung der Interface-Anbindung.

Parameter

[in] *InterfaceDll* — Dateiname der Interface-DLL mit Pfad

[in] *DataReceived* — Callback-Funktion, die beim Eintreffen von Daten aufgerufen wird. Die empfangenen Daten müssen anschließend über die Funktion **ReadAnswer** ausgelesen werden.

NULL, wenn asynchroner Datenempfang nicht verwendet wird.

[in] *TraceValuesReceived* — Callback-Funktion, die nach Auswertung empfangender Trace-Daten aufgerufen wird und die Trace-Daten übergibt.

NULL, wenn die Trace-Funktion nicht verwendet wird.

Rückgabe

eMomanprot_ok — Interface-DLL erfolgreich initialisiert

eMomanprot_error — Fehler beim Laden der Interface-DLL

Zur Verwendung der Callback-Funktionen müssen die jeweiligen Funktionszeiger deklariert sein:

```
typedef void (*tdmmProtDataCallback)(void);
typedef void (*tdmmProtTraceValuesCallback)(int nodeNr,
             unsigned int value[], int timecode);
```

Der Aufruf dieser Callback-Funktionen findet im Empfangs-Thread statt. Daher dürfen hier keine längeren Verarbeitungen vorgenommen werden.

Die Callback-Funktionen dienen nur zur Signalisierung des Datenempfangs. Die Weiterverarbeitung der empfangenen Daten, die in der Protokoll-Schicht zwischengespeichert sind, sollte in einem anderen Thread (z. B. dem Haupt-Thread der Anwendung) stattfinden.

Beispiel

```
tdmmProtDataCallback DataReceived;
tdmmProtTraceValuesCallback TraceValuesReceived;
std::string InterfaceDll = "Ixxat_vci3.dll";
eMomanprot ret = mmProtInitInterface((char*)InterfaceDll.c_str(),
     DataReceived, TraceValuesReceived);
```

5.2.2 CloseInterface

```
void mmProtCloseInterface (void)
```

Interface-Anbindung lösen und Speicher freigeben.

5.2.3 OpenCom

```
eMomanprot mmProtOpenCom ( int   port,
                           int   channel,
                           int   baud
                         )
```

Schnittstelle öffnen.

Parameter

[in] *port* Portnummer
COM: 0
USB: fortlaufend ab 1 für jedes USB-Gerät am ausgewählten Interface
CAN: fortlaufend ab 1 für jede CAN-Karte am ausgewählten Interface

[in] *channel* Kanalnummer
COM: COM-Nummer, z. B. 8 für COM8
USB: 0
CAN: 0 für Port mit nur einem Kanal, sonst fortlaufend ab 1

[in] *baud* Baudrate in Bit/s
COM: z. B. 9600
USB: 0
CAN: z. B. 250000

Rückgabe

eMomanprot_ok Schnittstelle erfolgreich geöffnet

eMomanprot_error Fehler beim Öffnen der Schnittstelle

Siehe Kap. 6.1, S. 53 zur Ermittlung der verfügbaren Ports und Kanäle.

Siehe Kap. 5.6.8, S. 52 zur Ermittlung der unterstützten Baudraten.

Beispiel
Ersten MC V3.x USB-Port öffnen:

```
eMomanprot ret = mmProtOpenCom(1, 0, 0);
```

5.2.4 CloseCom

```
void mmProtCloseCom (void)
```

Schnittstelle schließen.

5.2.5 ScanNode

```
int mmProtScanNode ( int          nodeNr,
                     const char** deviceName,
                     int&         deviceMode
                   )
```

Netzwerkknoten suchen.

Parameter

[in] *nodeNr* Knotennummer
–1: Broadcast

[out] *deviceName* Ausgelesener Gerätename

[out] *deviceMode* Geräteeigenschaft eDeviceMode

eDeviceMode_Faulhaber	FAULHABER Antrieb
eDeviceMode_Bootloader	FAULHABER Antrieb im Bootloader-Modus
eDeviceMode_AnyDrive	Nicht-FAULHABER Antrieb
eDeviceMode_AnyDevice	Anderes Gerät, kein Antrieb

Rückgabe

≥0 Knotennummer, wenn Knoten gefunden wurde

–1 Kein Knoten unter angegebener Knotennummer gefunden

Es wird geprüft, ob ein Knoten mit der angegebenen Knotennummer vorhanden ist. Falls nur ein Knoten mit einer unbekannten Knotennummer angeschlossen ist, kann die Knotennummer auch über einen Broadcast-Aufruf ermittelt werden.

Beispiel

```
std::string deviceName;
int deviceMode;
const char* name = NULL;
int foundNodeNr = mmProtScanNode (-1, &name, deviceMode);
if (name != NULL) deviceName = name;
```

5.2.6 LoadCommandSet

`int` **`mmProtLoadCommandSet (`**`int` *`cmdType`***`)`**

- Befehlssatz eines CAN-Gerätetyps laden, sofern es sich nicht um einen MC V3.x handelt.
- Aktuell geladenen Befehlssatz auslesen (*`eCmdType_LoadedCommandSet`*).

Parameter

[in] *`cmdType`*	Nummer des Gerätetyp-Befehlssatzes (eCmdType):	
	`eCmdType_LoadedCommandSet`	Geladenen Befehlssatz auslesen
	`eCmdType_Default`	Befehlssatz MC V3.x laden
	`eCmdType_MC3`	Befehlssatz MC V3.x laden
	`eCmdType_CO`	Befehlssatz MC V2.x CO laden
	`eCmdType_CF`	Befehlssatz MC V2.x CF laden

Rückgabe

eCommandSet_invalidCmdType	Ungültiger Übergabeparameter
eCommandSet_Unknown	Unbekannter Befehlssatz
eCommandSet_MC3RS	Befehlssatz MC V3.x RS232
eCommandSet_MC3USB	Befehlssatz MC V3.x USB
eCommandSet_MC3CAN	Befehlssatz MC V3.x CAN
eCommandSet_MC2RS	Befehlssatz MC V2.x RS
eCommandSet_MC2CAN	Befehlssatz MC V2.x CAN

5.2.7 SetDataCallback

`void` **`mmProtSetDataCallback (`**`tdmmProtDataCallback` *`DataReceived`***`)`**

Callback-Funktion zur Signalisierung asynchron eintreffender Daten einstellen, als Alternative zur Angabe bei **`InitInterface`** . Hiermit kann unabhängig von der Initialisierung des Interfaces zu einem beliebigen Zeitpunkt eine Callback-Funktion eingestellt oder auch gelöscht werden.

Parameter

[in] *`DataReceived`* Callback-Funktion, die beim Eintreffen von Daten aufgerufen wird. Die empfangenen Daten müssen anschließend über die Funktion **`ReadAnswer`** ausgelesen werden.

NULL, wenn der asynchrone Datenempfang ausgeschaltet werden soll.

Zur Verwendung der Callback-Funktion muss ein entsprechender Funktionszeiger deklariert sein:

```
typedef void (*tdmmProtDataCallback)(void);
```

Der Aufruf dieser Callback-Funktion findet im Empfangs-Thread statt. Daher dürfen hier keine längeren Verarbeitungen vorgenommen werden.

Die Callback-Funktion dient nur zur Signalisierung des Datenempfangs. Die Weiterverarbeitung der anschließend ausgelesenen Daten sollte in einem anderen Thread (z. B. dem Haupt-Thread der Anwendung) stattfinden.

5.2.8 SetTraceValuesCallback

`void` **`mmProtSetTraceValuesCallback (`**`tdmmProtTraceValuesCallback` *`TraceValuesReceived`***`)`**

Callback-Funktion zur Signalisierung asynchron eintreffender Trace-Daten einstellen, als Alternative zur Angabe bei **`InitInterface`** . Hiermit kann unabhängig von der Initialisierung des Interfaces zu einem beliebigen Zeitpunkt eine Callback-Funktion eingestellt oder auch gelöscht werden.

Parameter

[in] *`TraceValuesReceived`* Callback-Funktion, die nach Auswertung empfangender Trace-Daten aufgerufen wird und die Trace-Daten übergibt.

NULL, wenn die Trace-Funktion ausgeschaltet werden soll.

Zur Verwendung der Callback-Funktion muss ein entsprechender Funktionszeiger deklariert sein:

```
typedef void (*tdmmProtTraceValuesCallback)(int nodeNr,
             unsigned int value[], int timecode);
```

Der Aufruf dieser Callback-Funktion findet im Empfangs-Thread statt. Daher dürfen hier keine längeren Verarbeitungen vorgenommen werden.

Die Weiterverarbeitung der empfangenen Daten sollte in einem anderen Thread (z. B. dem Haupt-Thread der Anwendung) stattfinden.

5.3 Zugriff auf das Objektverzeichnis

Die Funktionen für den Zugriff auf das Objektverzeichnis stellen einfache Methoden für das synchrone Lesen und Schreiben von Objekten zur Verfügung. Diese Methoden werden von Motion Controllern der Familie MC V2.x RS nicht unterstützt.

Die zu übergebenden Werte für *index*, *subindex* und *len* können auf eine der folgenden Weisen ermittelt werden:

- Aus dem Kommunikationshandbuch oder dem Funktionshandbuch des jeweiligen Motion Controllers
- Über den Motion Manager:

  Im Objekt Browser des Motion Managers sind alle vom Gerät unterstützten Objekte aufgeführt. Aus der zugehörigen Statuszeile können der Wertebereich und der Datentyp für den Parameter *len* des ausgewählten Objekts abgelesen werden.

  Der Aufzeichnung im Terminal-Log des Motion Cockpits kann die Befehlsfolge entnommen werden, die für bestimmte Aktionen gesendet werden muss.

5.3.1 GetObj

```
int mmProtGetObj ( int  nodeNr,
                   int  index,
                   int  subIndex,
                   int& value
                 )
```

Integer-Objekteintrag aus Objektverzeichnis auslesen.

Parameter

[in] *nodeNr* Knotennummer

[in] *index* Index des Objekts

[in] *subIndex* Subindex des Objekts

[in] *value* Ausgelesener Integer-Wert

Rückgabe

eMomanprot_ok	Antwort erfolgreich empfangen
eMomanprot_error_timeout	Keine Antwort
eMomanprot_error_cmd	Fehlermeldung vom Gerät → CiA-Fehlercode wird über *value* zurückgegeben
eMomanprot_error	Anderer Fehler

Beispiel
Ist-Position von Knoten 1 abfragen:

```
int value;
eMomanprot ret = mmProtGetObj (1, 0x6064, 0x00, value);
```

5.3.2 GetInt64Obj

```
int mmProtGetInt64Obj ( int      nodeNr,
                        int      index,
                        int      subIndex,
                        _int64&  value
                      )
```

Objekte, die als Momancmd hinterlegt sind, als 64Bit-Integer aus dem Objektverzeichnis auslesen. Somit kann zwischen `unsigned int32` und `signed int32` unterschieden werden. Alle anderen Objekte werden analog zu **`GetObj`** als `signed int32` zurückgegeben.

Parameter

[in]	*nodeNr*	Knotennummer
[in]	*index*	Index des Objekts
[in]	*subIndex*	Subindex des Objekts
[out]	*value*	Ausgelesener Wert als Int64

Rückgabe

eMomanprot_ok	Antwort erfolgreich empfangen
eMomanprot_error_timeout	Keine Antwort
eMomanprot_error_cmd	Fehlermeldung vom Gerät → CiA-Fehlercode wird über *value* zurückgegeben
eMomanprot_error	Anderer Fehler

5.3.3 GetStrObj

```
int mmProtGetStrObj  ( int           nodeNr,
                       int           index,
                       int           subIndex,
                       const char**  value
                     )
```

String-Objekteintrag aus Objektverzeichnis auslesen.

Parameter

[in]	*nodeNr*	Knotennummer
[in]	*index*	Index des Objekts
[in]	*subIndex*	Subindex des Objekts
[out]	*value*	Ausgelesener String-Wert

Rückgabe

eMomanprot_ok	Antwort erfolgreich empfangen
eMomanprot_error_timeout	Keine Antwort
eMomanprot_error_cmd	Fehlermeldung vom Gerät → CiA-Fehlercode wird über *value* als hexadezimaler Wert zurückgegeben
eMomanprot_error	Anderer Fehler

Beispiel
Software-Version von Knoten 1 abfragen:

```
std::string version;
const char* ver = NULL;
eMomanprot ret = mmProtGetStrObj (1, 0x100A, 0x00, &ver);
if (ver != NULL) version = ver;
```

5.3.4 SetObj

```
int mmProtSetObj ( int           nodeNr,
                   int           index,
                   int           subIndex,
                   int           value,
                   int           len,
                   unsigned int& abortCode
                 )
```

Integer-Objekteintrag im Objektverzeichnis beschreiben.

Parameter

[in] *nodeNr* Knotennummer

[in] *index* Index des Objekts

[in] *subIndex* Subindex des Objekts

[in] *value* Neuer Wert

[in] *len* Datenlänge des Objekteintrags in Byte

[out] *abortCode* CiA-Fehlercode bei *eMomanprot_error_cmd*

Rückgabe

eMomanprot_ok	Wert erfolgreich übertragen
eMomanprot_error_timeout	Keine Antwort
eMomanprot_error_cmd	Fehlermeldung vom Gerät
eMomanprot_error	Anderer Fehler

Beispiel

Soll-Position = 1000 für Knoten 1 vorgeben:

```
unsigned int abortCode;
eMomanprot ret = mmProtSetObj (1, 0x607A, 0x00, 1000, 4, abortCode);
```

5.3.5 SetStrObj

```
int mmProtSetStrObj ( int           nodeNr,
                      int           index,
                      int           subIndex,
                      char*         value,
                      unsigned int& abortCode
                    )
```

String-Objekteintrag im Objektverzeichnis beschreiben.

Parameter

[in] *nodeNr* Knotennummer

[in] *index* Index des Objekts

[in] *subIndex* Subindex des Objekts

[in] *value* Neuer Wert

[out] *abortCode* CiA-Fehlercode bei *eMomanprot_error_cmd*

Rückgabe

eMomanprot_ok	Wert erfolgreich übertragen
eMomanprot_error_timeout	Keine Antwort
eMomanprot_error_cmd	Fehlermeldung vom Gerät
eMomanprot_error	Anderer Fehler

5.3.6 SetObjTimeout

```
void mmProtSetObjTimeout (int timeout)
```

Für Objektabfragen Timeout für Befehle setzen, die eine längere Ausführungszeit benötigen (z. B. Speichern, Wiederherstellen oder Rücksetzen). Standardmäßig ist der Timeout-Wert auf 500 ms eingestellt.

Parameter

[in] *timeout* ≥500: Timeout-Wert in ms
<500: Standardwert (500 ms)

5.3.7 GetAbortMessage

```
const char* mmProtGetAbortMessage (unsigned int abortCode)
```

Fehlermeldungstext des übergebenen Abortcodes auslesen.

Parameter

[in] *abortCode* Fehlercode

Rückgabe
Text der Abort-Meldung

5.4 Erweiterte Kommunikation

Die erweiterten Kommunikations-Funktionen stehen für Befehle ohne Objektverzeichnis-Bezug und für asynchronen Datenempfang zur Verfügung. Zusätzlich existieren Funktionen zur Analyse von Kommunikationsdaten, zur Auswertung der Kommunikations-Historie und zur Einstellung eines CAN-Nachrichtenfilters.

5.4.1 SendMotionCommand

```
bool mmProtSendMotionCommand  ( char* cmd,
                                int   nodeNr
                              )
```

ASCII-Motion-Befehl versenden.

Parameter

[in] *cmd* ASCII-Befehl

[in] *nodeNr* Knotennummer, falls nicht im Befehl enthalten

Rückgabe

true Befehl erfolgreich versendet

false Fehler beim Versenden

Als ASCII-Befehle können die in der Motion Manager Befehlsreferenz aufgeführten Befehle verwendet werden.

Bei Motion Controllern der Familie MC V2.x mit Schnittstelle RS können alle im Funktionshandbuch der jeweiligen Steuerung aufgeführten Befehle verwendet werden, die direkt zur Antriebssteuerung gesendet werden. Diese Befehle können auch bei Motion Controllern der Familie MC V2.x mit Schnittstelle CF verwendet werden, die hier über ein CAN-Telegramm versendet werden.

Auf eine Antwort wird nicht gewartet. Dazu steht die Funktion **`WaitAnswer`** zur Verfügung. Wenn bei **`InitInterface`** eine DataReceived-Callback-Funktion angegeben ist und asynchron eine Antwort eintrifft, wird die DataReceived-Callback-Funktion aufgerufen. Über **`ReadAnswer`** kann daraufhin die Antwort eingelesen werden.

5.4.2 SendCommand

```
bool mmProtSendCommand  ( int   nodeNr,
                          int   index,
                          int   subIndex,
                          int   dataLen,
                          int   data
                        )
```

Objekt-Befehl versenden.

Parameter

[in] *nodeNr* Knotennummer

[in] *index* Index des Daten-Objekts

[in] *subIndex* Subindex des Daten-Objekts

[in] *dataLen* Datenlänge und Befehlsart:

- 0: Abfragebefehl für Standard-Objekte (SDO Upload) oder Motion Manager Befehl ohne Daten (z. B. NMT, Device Control)
- –1: Abfragebefehl für lange Objekte (Segmented SDO Upload)
- 1…4: Sendebefehl für Standard-Objekte (SDO Download)
- >4: Sendebefehl für lange Objekte (Segmented SDO Download)

[in] *data* Zu versendender Daten-Wert oder Zeiger auf Datenblock

0, wenn *dataLen* = 0

Rückgabe

true Befehl erfolgreich versendet

false Fehler beim Versenden

Mit dieser Funktion können Parameter im Objektverzeichnis gelesen oder beschrieben werden, ohne auf die Antwort zu warten. Über einen nachfolgenden Aufruf von **WaitAnswer** kann die Antwort eingelesen werden (z. B. Auslesen eines Datenpuffers). Wenn bei **InitInterface** eine DataReceived-Callback-Funktion angegeben ist und asynchron eine Antwort eintrifft, wird die DataReceived-Callback-Funktion aufgerufen. Über **ReadAnswer** kann daraufhin die Antwort eingelesen werden.

Für das synchrone Lesen und Schreiben von Objekten stehen die Funktionen in Kap. 5.3, S. 23 zur Verfügung. Für das Senden von Datenpuffer-Objekten sollte die Funktion **SendBuffer** verwendet werden.

Zusätzlich zu den im Gerät definierten Objekten stehen auf Index 0x0000 spezielle Motion Manager Befehle zur Verfügung, die für Standardaufgaben, NMT und Device Control verwendet werden können (siehe `enum eMomancmd` in **Momancmd.h**).

5.4.3 SendBuffer

```
bool mmProtSendBuffer ( int   nodeNr,
                        int   index,
                        int   subIndex,
                        int   dataLen,
                        char* dataBuffer
                      )
```

Puffer-Inhalt oder String über Objekt-Befehl versenden.

Parameter

[in] *nodeNr* Knotennummer

[in] *index* Index des Daten-Objekts

[in] *subIndex* Subindex des Daten-Objekts

[in] *dataLen* Datenlänge in Byte

[in] *dataBuffer* Zeiger auf zu versendenden Datenblock

Rückgabe

true Befehl erfolgreich versendet

false Fehler beim Versenden

Mit dieser Funktion können String- und Datenpufferobjekte im Objektverzeichnis beschrieben werden.

Bei fehlgeschlagenem Funktionsaufruf können bei Bedarf über **`WaitAnswer`** Fehlercode und Fehlertext ausgelesen werden.

5.4.4 SendTelegram

```
bool mmProtSendTelegram ( int   id,
                          char* data,
                          int   len
                        )
```

Beliebiges Telegramm versenden.

Auf die angegebene COB-ID (CAN) oder Knotennummer (RS232, USB) werden direkt die angegebenen Daten versendet. Der Telegrammaufbau ist im Kommunikationshandbuch beschrieben.

Wird von Motion Controllern der Familie MC V2.x RS nicht unterstützt.

Parameter

[in] *id* COB-ID oder Knotennummer

[in] *data* Datenfeld

[in] *len* Länge des Datenfelds

Rückgabe

true Telegramm erfolgreich versendet

false Fehler beim Versenden

5.4.5 WaitAnswer

```
eMomanprot mmProtWaitAnswer ( int          timeout,
                              int          answ,
                              const char** answData
                            )
```

Auf Antwort vom Gerät warten.

Parameter

[in]	*timeout*	Timeout-Zeit in ms
[in]	*answ*	Antwort-Modus (eWaitMode)
[out]	*answData*	Antwort auf zuvor gesendeten Befehl Bei Rückgabe von *eMomanprot_error_cmd* enthält *answData* einen Fehlertext.

Rückgabe

eMomanprot_ok	Antwort erfolgreich empfangen
eMomanprot_error_timeout	Keine Antwort
eMomanprot_error_cmd	Fehlermeldung vom Gerät
eMomanprot_error_param	Ungültiger Übergabeparameter
eMomanprot_error	Anderer Fehler
>4	Länge des empfangenen Binärdaten-Pakets

Diese Funktion steht für die synchrone Kommunikation mit den Funktionen **SendMotionCommand**, **SendCommand** und **SendBuffer** zur Verfügung.

Der Übergabeparameter *answ* kann einen der folgenden Werte erhalten:

eWaitMode_Int	Warten auf eine Integer-Antwort
eWaitMode_String	Warten auf eine String-Antwort
eWaitMode_BinData	Warten auf einen Binärdaten-Puffer > 4 Byte
eWaitMode_noAsync_noAck	Warten auf MC V2.x RS Antwort ohne Async. und Ack.
eWaitMode_noAck	Warten auf MC V2.x RS Antwort ohne Acknowledge
eWaitMode_noAsync	Warten auf MC V2.x RS Antwort ohne Async.
eWaitMode_someAsync	Warten auf MC V2.x RS Antwort mit bestimmten Async.
eWaitMode_Bin	Warten auf MC V2.x RS Binär-Antwort Für das Warten auf Trace-Daten, werden immer 9 Byte zurückgegeben: Die ersten 4 Byte für Trace-Parameter 1, die nächsten 4 Byte für Trace-Parameter 2 und abschließend 1 Byte Timecode.
>eWaitMode_Id	Warten auf Antwort mit bestimmtem Identifier (COB-ID)

Beispiel
Ist-Position von einem MC V2.x auslesen:

```
std::string position;
if (mmProtSendMotionCommand ("POS", 0) == true) {
      const char* data = NULL;
      eMomanprot ret = mmProtWaitAnswer (1000, eWaitMode_String, data);
      if (data != NULL) position = data;
}
```

5.4.6 ReadAnswer

```
eMomanprot mmProtReadAnswer ( const char**  answData,
                              int&          nodeNr,
                              const char**  cmdString,
                              const char**  receiveTelegram
                            )
```

Empfangene Nachrichten auslesen.

Nach einer Empfangsbenachrichtigung (DataReceived-Callback) können hiermit die im Ringpuffer zwischengespeicherten Nachrichten ausgelesen werden.

Parameter

[out] *answData*	Als String interpretierte Antwort-Daten
[out] *nodeNr*	Knotennummer der empfangenen Nachricht
[out] *cmdString*	Zugehöriger Befehls-String
[out] *receiveTelegram*	Empfangstelegramm

Rückgabe

eMomanprot_ok	Antwort erfolgreich empfangen
eMomanprot_error_cmd	Fehlermeldung vom Gerät
eMomanprot_error_emcy	Emergency-Fehler vom Gerät
eMomanprot_error	Anderer Fehler
eMomanprot_noData	Keine Daten zum Auslesen verfügbar

ReadAnswer sollte nicht direkt innerhalb der DataReceived-Callback-Funktion aufgerufen werden, damit der Empfangs-Thread nicht ausgebremst wird.

Die Signalisierung des Datenempfangs und das Einlesen der Daten müssen voneinander entkoppelt sein. Zur Entkopplung kann **ReadAnswer** z. B. in einem eigenen Thread (Event-Signalisierung), in einem Windows-Messagehandler (Signalisierung über Post-Message) oder innerhalb eines Timer-Events aufgerufen werden.

Über die Hilfsfunktionen **DecodeAnswStr** und **DecodeCmdStr** können die empfangenen Daten weiter untersucht werden.

Beispiel

```
std::string data, cmd;
int statusword;
const char* answData = NULL;
const char* cmdString = NULL;
const char* receiveTelegram = NULL;
int nodeNr;

eMomanprot ret = mmProtReadAnswer (&answData, nodeNr, &cmdString, &receiveTele-
gram);

if (ret != eMomanprot_noData) {
      if (answData != NULL) data = answData;
      if (cmdString != NULL) cmd = cmdString;
      _int64 value;
      if (mmProtDecodeAnswStr (answData, value) == eDecoded_Statusword) {
            statusword = (int)value;
      }
}
```

5.4.7 DecodeAnswStr

```
eDecoded mmProtDecodeAnswStr ( const char** answStr,
                                _int64&      value
                              )
```

Den übergebenen **ReadAnswer**-Antwort-String nach bestimmten Eigenschaften dekodieren.

Parameter

[in] *answStr* Antwort-String aus **ReadAnswer**

[out] *value* Im Antwort-String enthaltener Wert

Rückgabe

eDecoded_none	*answStr* ist nicht dekodiert
eDecoded_Bootup	*answStr* ist eine Bootup-Nachricht
eDecoded_NMT	*answStr* ist eine NMT-Nachricht
eDecoded_NMTRequest	*answStr* ist eine NMT-Request-Antwort
eDecoded_Heartbeat	*answStr* ist eine Heartbeat-Nachricht
eDecoded_Statusword	*answStr* ist eine Statusword-Nachricht

5.4.8 DecodeCmdStr

```
eDecoded mmProtDecodeCmdStr ( const char*    cmdStr,
                              int&           nodeNr,
                              int&           index,
                              int&           subIndex,
                              const char**   valueStr
                            )
```

Den übergebenen **ReadAnswer**-Befehls-String dekodieren. Falls es ein Objekt-Befehl ist, werden Knotennummer, Index, Subindex und SOBJ-Wert zurückgeben.

Parameter

[in]	*cmdStr*	Befehls-String
[out]	*nodeNr*	Knotennummer
[out]	*index*	Objekt-Index
[out]	*subIndex*	Objekt-Subindex
[out]	*valueStr*	Zu schreibender Wert bei SOBJ-Befehl

Rückgabe

eDecoded_none	*cmdStr* ist nicht dekodiert
eDecoded_SOBJ	*cmdStr* ist ein SOBJ-Befehl
eDecoded_GOBJ	*cmdStr* ist ein GOBJ-Befehl

5.4.9 CheckMotionCommand

`int` **`mmProtCheckMotionCommand`** `(char*` *`cmd`*`)`

Prüfen, ob der übergebene ASCII-Befehl mit einer bestimmten Aktion verbunden ist.

Parameter

[in] *`cmd`* ASCII-Befehl

Rückgabe

eCheckCommand_noAction	Keine spezielle Aktion
eCheckCommand_State	Befehl kann den NMT-, Device- oder Opmode-Zustand ändern
eCheckCommand_SwitchOn	Befehl schaltet den Motor ein
eCheckCommand_Save	Befehl ist ein SAVE-Befehl
eCheckCommand_Restore	Befehl ist ein RESTORE-Befehl
eCheckCommand_Reset	Befehl ist ein RESET-Befehl

5.4.10 CheckCommand

```
eCheckCommand mmProtCheckCommand ( int  index,
                                   int  subIndex,
                                   int  dataLen,
                                   int  data
                                 )
```

Prüfen, ob mit dem übergebenen Objekt-Befehl eine bestimmte Aktion verbunden ist.

Parameter

[in] *index* Index des Daten-Objekts

[in] *subIndex* Subindex des Daten-Objekts

[in] *dataLen* Datenlänge in Byte

[in] *data* Zu versendender Datenwert

Rückgabe

eCheckCommand_noAction	Keine spezielle Aktion
eCheckCommand_State	Befehl kann den NMT-, Device- oder Opmode-Zustand ändern
eCheckCommand_SwitchOn	Befehl schaltet den Motor ein
eCheckCommand_Save	Befehl ist ein SAVE-Befehl
eCheckCommand_Restore	Befehl ist ein RESTORE-Befehl
eCheckCommand_Reset	Befehl ist ein RESET-Befehl

5.4.11 GetCommunicationHistory

```
bool mmProtGetCommunicationHistory ( const char**   timestamp,
                                     eHistoryState& state,
                                     const char**   data,
                                     const char**   telegram,
                                     eHistoryError& error
                                   )
```

Die in der Protokoll-Schicht zwischengespeicherte Kommunikations-Historie auslesen.

Kann in einer Schleife ausgelesen werden, solange *true* zurückgegeben wird.

Parameter

[out] *timestamp* Versende- bzw. Empfangs-Zeitstempel

[out] *state* Status des Telegramms:

eHistoryState_SendData	Versendete Daten
eHistoryState_ReceiveWaitData	Empfangene erwartete Daten
eHistoryState_ReceiveData	Asynchron empfangene Daten
eHistoryState_Message	Informative Meldung

[out] *data* Kommando / interpretierte Empfangsdaten

[out] *telegram* Sende- / Empfangs-Telegramm

[out] *error* Fehlercode:

eHistoryError_Ok	Kein Fehler
eHistoryError_SendError	Fehler beim Senden
eHistoryError_ReceiveError	Fehler empfangen
eHistoryError_ReceiveTimeout	Timeout
eHistoryError_ReceiveEmcy	Emergency-Fehler empfangen

Rückgabe

true Puffer-Inhalt ausgelesen

false Puffer leer

5.4.12 GetSendTelegram

```
void mmProtGetSendTelegram ( const char** sendTelegram,
                             const char** cmdString
                           )
```

Zuletzt übertragenes Sendetelegramm auslesen.

Parameter

[out] *sendTelegram* Sendetelegramm

[out] *cmdString* Sendebefehl

5.4.13 SetupMessageFilter

```
void mmProtSetupMessageFilter ( int nodeNr,
                                int activated,
                                int cobId,
                                int cobIdCount
                              )
```

Einen CAN-Nachrichtenfilter aktivieren.

Mit dem CAN-Nachrichtenfilter werden asynchrone CAN-Nachrichten ausgefiltert. Bei aktivem Filter können COB-ID-Ausnahmen angegeben werden, die weiterhin durchgelassen werden.

Parameter

[in] *nodeNr* Knotennummer, deren Telegramme immer angezeigt werden sollen

[in] *activated* 0: Nachrichtenfilter deaktiviert
1: Fremde Nachrichten bis auf Ausnahmen ausfiltern
2: Eigene TxPDOs bis auf Ausnahmen ausfiltern
3: Fremde Nachrichten und eigene TxPDOs ausfiltern

[in] *cobId* Array mit COB-ID-Ausnahmen

[in] *cobIdCount* Anzahl der COB-ID-Ausnahmen

5.5 Datenaufzeichnung

Mit den Funktionen zur Datenaufzeichnung können interne Geräte-Parameter kontinuierlich (Logger) oder gepuffert (Recorder) aufgezeichnet und ausgelesen werden.

Für die Verwendung der Trace-Funktion muss bei der Initialisierung eine Callback-Funktion angegeben werden, über die die aufgenommenen Daten asynchron an die Anwendung übergeben werden.

5.5.1 SetupTrace

```
eMomanprot mmProtSetupTrace  ( STraceSetup&    TraceSetup,
                               STraceTrigger&  TraceTrigger
                             )
```

Trace konfigurieren.

Parameter

[in] *TraceSetup*	Einstellung der Trace-Kanäle
[in] *TraceTrigger*	Triggereinstellungen

Rückgabe

eMomanprot_ok	Erfolgreich konfiguriert
eMomanprot_error	Konfigurationsfehler

Strukturen
Für die Trace-Konfiguration sind die Strukturen `STraceSetup` und `STraceTrigger` definiert:

```
typedef struct {
  bool run;           /*!< run or stop tracing */
  int  nodeNr;        /*!< node number to be traced */
  int  chan;          /*!< 0: default channel, 1: logger uses CAN-PDOs */
  int  mode;          /*!< 0: logger(single requests), 1: recorder(buffered) */
  int  source[4];     /*!< trace parameter: OD Index/Subindex */
  int  sourceLen[4]; /*!< data length in byte; 0: source not used */
  int  sourceType[4];/*!< 0: default; CAN logger: cobId of trace pdo;
                        -1: deactivated */
  int  buffer;        /*!< buffer size (number of samples to be recorded) */
  int  sampletime     /*!< sampletime in recorder mode */
} STraceSetup;
```

```
typedef struct {
  int  mode;         /*!< 0: no trigger, 1: singleshot, 2: retriggered */
  int  source;       /*!< trigger parameter: OD Index/Subindex */
  int  sourceType;   /*!< TrigType: 0 = OD */
  int  threshold;    /*!< trigger if source value > or < threshold value */
  int  edge          /*!< 0: rising (>), 1: falling (<) */
  int  delay         /*!< triggerdelay */
} STraceTrigger;
```

Die aufgeführten Trace- und Trigger-Einstellungen stehen nicht bei allen Geräten zur Verfügung (siehe Funktionshandbuch zum jeweiligen Gerät).

Beispiel Logger MC V2.x RS

```
STraceTrigger traceTrigger;          //not used
STraceSetup traceSetup;
traceSetup.run = true;
traceSetup.source[0] = 200;          //Trace parameter 1: actual position
traceSetup.source[1] = 255;          //Trace parameter 2: not used
traceSetup.sampletime = 0;
eMomanprot ret = mmProtSetupTrace(traceSetup, traceTrigger);
```

Beispiel Logger MC V3.x RS/USB

```
STraceTrigger traceTrigger;          //not used
STraceSetup traceSetup;
traceSetup.run = true;
traceSetup.nodeNr = 1;
traceSetup.chan = 0;                 //Default channel
traceSetup.mode = 0;                 //Logger
traceSetup.source[0] = 0x606400;     //Trace parameter 1: actual position
traceSetup.source[1] = 0x606C00;     //Trace parameter 2: actual speed
traceSetup.source[2] = 0x607800;     //Trace parameter 3: actual current
                                     consumption
traceSetup.source[3] = 0;            //Trace parameter 4: not used
traceSetup.sourceLen[0] = 4;
traceSetup.sourceLen[1] = 4;
traceSetup.sourceLen[2] = 2;
traceSetup.sourceLen[3] = 0;
traceSetup.sourceType[0] = 0;
traceSetup.sourceType[1] = 0;
traceSetup.sourceType[2] = 0;
traceSetup.sourceType[3] = -1;
eMomanprot ret = mmProtSetupTrace(traceSetup, traceTrigger);
```

Beispiel Logger MC V3.x CAN

```
STraceTrigger traceTrigger;          //not used
STraceSetup traceSetup;
traceSetup.run = true;
traceSetup.nodeNr = 1;
traceSetup.chan = 1;                 //Logger uses CAN-PDOs
traceSetup.mode = 0;                 //Logger
traceSetup.source[0] = 0x606400;     //Trace parameter 1: actual position
traceSetup.source[1] = 0x606C00;     //Trace parameter 2: actual speed
traceSetup.source[2] = 0;            //Trace parameter 3: not used
traceSetup.source[3] = 0;            //Trace parameter 4: not used
traceSetup.sourceLen[0] = 4;
traceSetup.sourceLen[1] = 4;
traceSetup.sourceLen[2] = 0;
traceSetup.sourceLen[3] = 0;
traceSetup.sourceType[0] = 0x481;    //CobId of trace pdo for Source0
traceSetup.sourceType[1] = 0x481;    //CobId of trace pdo for Source1
traceSetup.sourceType[2] = -1;       //Deactivated
traceSetup.sourceType[3] = -1;       //Deactivated

eMomanprot ret = mmProtSetupTrace(traceSetup, traceTrigger);
```

Beispiele Recorder MC V3.x

```
STraceTrigger traceTrigger;
traceTrigger.mode = 0;                  //No trigger
STraceSetup traceSetup;
traceSetup.run = true;
traceSetup.nodeNr = 1;
traceSetup.chan = 0;                    //Default channel
traceSetup.mode = 1;                    //Recorder
traceSetup.source[0] = 0x606400;        //Trace parameter 1: actual position
traceSetup.source[1] = 0x606C00;        //Trace parameter 2: actual speed
traceSetup.source[2] = 0;               //Trace parameter 3: not used
traceSetup.source[3] = 0;               //Trace parameter 4: not used
traceSetup.sourceLen[0] = 4;
traceSetup.sourceLen[1] = 4;
traceSetup.sourceLen[2] = 0;
traceSetup.sourceLen[3] = 0;
traceSetup.sourceType[0] = 0;
traceSetup.sourceType[1] = 0;
traceSetup.sourceType[2] = -1;
traceSetup.sourceType[3] = -1;
traceSetup.buffer = 512;
traceSetup.sampletime = 1;

eMomanprot ret = mmProtSetupTrace(traceSetup, traceTrigger);
```

5.5.2 RequestTrace

`eTraceRequest` **`mmProtRequestTrace (`**`int` *`mode`***`)`**

Trace-Anforderung versenden.

Parameter

[in] *`mode`* Request-Mode
0: Standard (aufgezeichnete Daten sofort einlesen)
1: Im Recorder-Mode nur Statusüberprüfung
2: Im Recorder-Mode aufgezeichnete Daten einlesen

Rückgabe

eTraceRequest_inactive	Trace nicht aktiviert
eTraceRequest_sent	Trace-Request versendet
eTraceRequest_stateWaitResponse	Recorder erwartet Antwort auf Trace-Request
eTraceRequest_stateWaitForTrigger	Recorder wartet auf erfüllte Triggerbedingung
eTraceRequest_stateRecordingInProgress	Recorder-Aufzeichnung läuft
eTraceRequest_stateRecordingFinished	Recorder-Aufzeichnung abgeschlossen
eTraceRequest_errorReadBuffer	Fehler beim Auslesen des Recorder-Puffers

Logger
Durch Ausführung von **`RequestTrace(`***`0`***`)`** wird eine Einzeldatenanforderung an das Gerät gesendet. Darauf überträgt dieses Gerät sofort die konfigurierten Daten zum aktuellen Zeitpunkt. Die Daten werden dann an die vorgegebene TraceValuesReceived-Callback-Funktion weitergegeben.

Mit der TraceValuesReceived-Callback-Funktion werden folgende Werte zurückgegeben:

- Knotennummer des angesprochenen Geräts
- Array mit 4 Werten (unsigned int) der konfigurierten Quellen zum Einlesezeitpunkt
- Timecode als Zeitdifferenz zum letzten Einlesezeitpunkt in Millisekunden

Nach Verarbeitung der eingetroffenen Daten kann **`RequestTrace(`***`0`***`)`** erneut aufgerufen werden.

Recorder (nur MC V3.x)
Durch Ausführung von **`RequestTrace(`***`0`***`)`** oder **`RequestTrace(`***`1`***`)`** wird eine Recorder-Anforderung an das Gerät gesendet. Sobald die eingestellte Triggerbedingung erfüllt ist, zeichnet das Gerät die konfigurierten Daten auf.

- Mit *`mode`* `= 0` werden die aufgezeichneten Daten eingelesen, sobald sie verfügbar sind. Gleichzeitig wird *eTraceRequest_stateRecordingFinished* zurückgegeben.
- Mit *`mode`* `= 1` wird nur *eTraceRequest_stateRecordingFinished* zurückgegeben. Die Daten werden mit Ausführung von **`RequestTrace(`***`2`***`)`** eingelesen.

`RequestTrace(`*`0`***`)`** oder **`RequestTrace(`***`1`***`)`** muss in regelmäßigen Abständen aufgerufen werden, um Informationen über den aktuellen Trace-Request-Status zu erhalten und die aufgezeichneten Daten anzufordern, sobald sie verfügbar sind.

Nach dem Einlesen werden die Daten hintereinander an die vorgegebene TraceValues-Received-Callback-Funktion weitergegeben.

Mit der TraceValuesReceived-Callback-Funktion werden folgende Werte zurückgegeben:

- Knotennummer des angesprochenen Geräts
- Array mit 4 Werten (unsigned int) der konfigurierten Quellen zum Einlesezeitpunkt
- Timecode mit Wert der eingestellten Sample-Time

Die Anzahl Aufrufe der Callback-Funktion entspricht der eingestellten Größe des Trace-Puffers.

Nach der Übertragung des Trace-Puffers wird die Aufzeichnung im Gerät sofort wieder aktiviert (Trigger Enable). Über weitere Aufrufe von **`RequestTrace(`**`0`**`)`** oder **`RequestTrace(`**`1`**`)`** kann die Verfügbarkeit neuer Daten geprüft werden.

5.6 Verbindungseinstellungen

Mit den Funktionen zur Verbindungseinstellung können die Kommunikationseinstellungen angeschlossener Geräte ausgelesen und geändert werden. Bei unbekannter Kommunikationseinstellung kann eine Verbindung zu angeschlossenen Geräten gesucht werden.

5.6.1 NetworkService

```
eMomanprot mmProtNetworkService ( int   mode,
                                  int   vendorId,
                                  int   productCode,
                                  int   revisionNumber,
                                  int   serialNumber
                                )
```

Netzwerkknoten identifizieren und adressieren, auch wenn deren Knotennummer nicht bekannt oder noch nicht gesetzt ist.

Bei CANopen wird hierfür das LSS-Protokoll nach CiA 305 verwendet.

Parameter

[in]	*mode*	Modus des Netzwerk-Services: 0: LSS switch normal mode 1: LSS switch config mode global 2: LSS switch config mode selective 3: LSS identify remote slave 4: Select nodeNr (in param serialNumber)
[in]	*vendorId*	CAN-ID des Herstellers (0x1018.01)
[in]	*productCode*	Identifikationsnummer des Produkts (0x1018.02)
[out]	*revisionNumber*	Versionsnummer des Produkts (0x1018.03)
[out]	*serialNumber*	Seriennummer (0x1018.04) / Knotennummer

Rückgabe

>0	Anzahl der gefundenen Knoten
eMomanprot_ok	Funktion erfolgreich ausgeführt
eMomanprot_error	Fehler bei der Ausführung der Funktion

Für die Angaben *vendorId*, *productCode* und *revisionNumber* siehe Funktions- oder Kommunikationshandbuch des jeweiligen Geräts. Die *serialNumber* ist in der Regel am Produkt aufgedruckt.

5.6.2 GetCommunicationSettings

```
bool mmProtGetCommunicationSettings (int& nodeNr,
                                     int& baudrate
                                    )
```

Kommunikationseinstellungen des über **NetworkService** adressierten Knotens auslesen.

Parameter

[out] *nodeNr* Aktuell eingestellte Knotennummer

[out] *baudrate* Aktuell eingestellte Baudrate

Rückgabe

true Kommunikationseinstellungen erfolgreich ausgelesen

false Fehler beim Auslesen der Kommunikationseinstellungen

5.6.3 ChangeNodeNr

```
bool mmProtChangeNodeNr (int nodeNr)
```

Knotennummer des über **NetworkService** adressierten Knotens ändern.

Parameter

[in] *nodeNr* Neue Knotennummer

Rückgabe

true Knotennummer erfolgreich geändert

false Änderung der Knotennummer nicht möglich

5.6.4 ChangeBaudrate

```
bool mmProtChangeBaudrate ( int  baudrate,
                            int  activate
                          )
```

Baudrate des über **NetworkService** adressierten Knotens ändern.

Parameter

[in] *baudrate* Neu einzustellende Baudrate in Bit/s
0: Autobaud

[in] *activate* 0: Baudrate erst später aktivieren (Aufruf mit *activate* = 2)
1: Baudrate sofort aktivieren
2: Zuvor eingestellte Baudrate aktivieren (Parameter *baudrate* hier unbenutzt)

Rückgabe

true Baudrate erfolgreich geändert

false Änderung der Baudrate nicht möglich

5.6.5 Connect

```
int mmProtConnect ( int&  baudrate,
                    bool  tryBaudrates
                  )
```

Verbindung zu einem Knoten kontrollieren oder herstellen.

Wenn noch keine Verbindung hergestellt wurde, wird ein Broadcast-Verfahren verwendet, um die Knotennummer eines angeschlossenen Geräts zu ermitteln.

Parameter

[in, out] *baudrate* In: Aktuelle Baudrate
Out: Ermittelte Baudrate in Bit/s, wenn *tryBaudrates* = true

[in] *tryBaudrates* True: Baudraten durchprobieren, wenn keine Verbindung möglich ist

Rückgabe

≥0 Erste verbundene Knotennummer (255: unkonfiguriert)

–1 Keine Verbindung gefunden

5.6.6 FindConnection

```
int mmProtFindConnection (int   scanMin,
                          int   scanMax,
                          int&  baudrate
                         )
```

Verbindung zu einem RS232-Knoten suchen.

Falls mit **Connect** über Broadcast keine RS232-Verbindung hergestellt werden konnte, liegt vielleicht ein RS232-Netzwerk vor, bei dem verschiedene Knotennummern mit unterschiedlicher Baudrate durchprobiert werden können.

Parameter

[in]	*scanMin*	Kleinste Knotennummer
[in]	*scanMax*	Größte Knotennummer
[out]	*baudrate*	Ermittelte Baudrate in Bit/s

Rückgabe

≥0	Erste verbundene Knotennummer (255: unkonfiguriert)
–1	Keine Verbindung gefunden

5.6.7 UnconfiguredSlavesCount

```
int mmProtUnconfiguredSlavesCount (void)
```

Anzahl der unkonfigurierten Knoten im Netzwerk auslesen.

Bei CANopen wird hierfür das LSS-Protokoll nach CiA 305 verwendet. Bei USB und RS232 wird immer 0 zurückgegeben.

Rückgabe
Anzahl der unkonfigurierten Knoten.

5.6.8 SupportedBaudratesList

```
const unsigned int* mmProtSupportedBaudratesList (int& count)
```

Liste der mit diesem Protokoll unterstützten Baudraten auslesen.

Parameter

[out] *count* Anzahl der unterstützten Baudraten

Rückgabe

Zeiger auf die konstante Baudratenliste.

Beispiel

```
int count = 0;
const unsigned int* baudList = NULL;
baudList = mmProtSupportedBaudratesList (count);
if (baudList != Null) {
      unsigned int baudrate;
      for (int i = 0; i < count; i++) baudrate = baudList[i];
}
```

6 Ports und Kanäle

Alle Interface-DLLs stellen zwei allgemeine Funktionen für den Zugriff auf Interface-Informationen zur Verfügung:

- **`mmIntfEnumPorts()`**

  Gibt eine Liste verfügbarer Ports und Kanäle inklusive der Port-Informationen des geladenen Interfaces zurück.

- **`mmIntfIsPortAvailable()`**

  Prüft, ob der angegebene Port/Kanal verfügbar oder bereits in Benutzung ist.

Die beiden Funktionen sind in der Datei **Momanintf.h** definiert.

6.1 EnumPorts

```
int mmIntfEnumPorts  ( const char**  portList,
                       const char**  chanList,
                       const char**  deviceInfoList
                     )
```

Verfügbare Ports und Kanäle des geladenen Interfaces auslesen.

Für jedes über eine Interface-DLL registrierte Interface (USB-Gerät, CAN-Karte) wird eine fortlaufende Port-ID vergeben. Über diese Port-ID kann bei Verwendung mehrerer USB-Geräte oder CAN-Karten dieses Interface-Typs wieder auf das dort angeschlossene Gerät zugegriffen werden. Bei COM ist die Port-ID immer 0. Die Kanalnummer bezeichnet bei COM den Kommunikationskanal (z. B. 5 für COM5), bei CAN-Karten mit mehreren Bus-Anschlüssen ist dies die Nummer des jeweiligen Kanals. Bei CAN-Karten mit nur einem Bus-Anschluss und bei USB ist die Kanalnummer 0.

Port-ID und Kanalnummer werden als Parameter der Protokoll-Funktion **`OpenCom`** benötigt.

Parameter

[out] *portList* — Liste der gefundenen Port-IDs (durch Kommas getrennt).

Hat ein Port mehrere Kanäle, wird die Port-ID für jeden Kanal angegeben.

Beispiel: "1,2,2" (Port 1 mit einem Kanal, Port 2 mit zwei Kanälen)

[out] *chanList* — Liste der Kanalnummern für jeden Port (durch Kommas getrennt).

0: Nur ein Kanal an diesem Port (keine Kanalnummer)

>0: Kanalnummer dieses Ports.

Beispiel: "0,1,2" (Port 1 mit einem Kanal, Port 2 mit Kanal 1 und 2)

[out] *deviceInfoList* — Liste der Geräteinformationen (Interface-Name, Seriennummer) für jeden Port (durch Kommas getrennt).

Rückgabe

Anzahl der gefundenen Kanäle.

6.2 IsPortAvailable

```
eMomanIntf mmIntfIsPortAvailable ( int port,
                                   int chan
                                 )
```

Prüfen, ob angegebener Port/Kanal verfügbar oder bereits in Benutzung ist.

Parameter

[in] *port* Port-ID

[in] *chan* Kanalnummer

Rückgabe

eMomanintf_not_avail	Port bzw. Kanal nicht vorhanden
eMomanintf_avail	Port vorhanden und verfügbar
eMomanintf_avail_inuse	Port vorhanden aber bereits in Benutzung

7 Alternative Programmiermöglichkeiten

Neben der Möglichkeit, die FAULHABER Motion Controller über die hier beschriebene MomanLib-API zu programmieren, können auch Standard-Methoden der jeweiligen Programmiersprache zur Kommunikation verwendet werden. Bei der Verwendung von Standard-Methoden gibt es keine Einschränkungen bezüglich Funktionalität und Performance. Zusätzlich entfällt die Abhängigkeit von weiteren Fremdkomponenten.

Der Motion Manager zeigt im Terminal-Fenster die Kommunikations-Historie mit dem Inhalt jedes gesendeten und empfangenen Telegramms an. Hierüber kann für jeden Befehl das gesendete Telegramm abgelesen und in ein eigenes Programm übernommen werden.

7.1 RS232

Bei den meisten Programmiersprachen ist die Unterstützung für die RS232-Schnittstelle standardmäßig vorhanden. Das zu verwendende Kommunikationsprotokoll ist im Kommunikationshandbuch des jeweiligen Controllers beschrieben.

- Motion Controller der Familie MC V2.x werden über einfache ASCII-Befehle mit einem abschließenden Carriage Return angesprochen. Diese Befehle können direkt Zeichen für Zeichen auf die RS232-Schnittstelle gesendet werden. Antworten werden auf die gleiche Weise eingelesen.
- Motion Controller der Familie MC V3.x werden über Binärtelegramme angesprochen. Zu beachten ist, dass außer den zu übertragenden Daten die Telegrammlänge und eine zu berechnende Checksumme im Telegramm enthalten sind. Der Algorithmus zur Berechnung der Checksumme ist im Kommunikationshandbuch des jeweiligen Controllers beschrieben.

7.1.1 C/C++

In C/C++ können Funktionen aus der Windows-API (`CreateFile()`, `WriteFile()`, `ReadFile()`) verwendet werden. Alternativ gibt es eine Reihe von kommerziellen und kostenlosen Libraries für die serielle Kommunikation.

7.1.2 Delphi

In Delphi können Funktionen aus der Windows-API (`CreateFile()`, `WriteFile()`, `ReadFile()`) verwendet werden. Alternativ gibt es eine Reihe von kommerziellen und kostenlosen Libraries für die serielle Kommunikation.

7.1.3 C#

In C# kann ein `System.IO.Ports.SerialPort` erzeugt werden, der dann entweder über das Interface `System.IO.Stream` oder die Klassenfunktionen `Read` und `Write` verwendet werden kann. Durch die Verwendung des Events `DataReceived` muss nicht ständiges Polling verwendet werden und wird bei strengen Performance-Anforderungen empfohlen.

7.1.4 LabVIEW

LabVIEW stellt auf der VISA-Palette "Seriell" VIs zur Kommunikation über die serielle Schnittstelle zur Verfügung.

7.2 USB

Bei Verwendung eines USB-to-Serial-Adapters können alle Controller mit RS232-Schnittstelle wie in Kap. 7.1, S. 55 beschrieben programmiert werden.

7.2.1 Virtueller COM-Port unter Windows 10

Unter Windows 10 kann ein FAULHABER Motion Controller der Familie MC V3.x alternativ zur direkten USB-Anbindung auch über einen virtuellen COM-Port angesprochen werden. Hierfür muss über den Windows-Gerätemanager die Treiberanbindung geändert werden:

1. Treibersoftware aktualisieren.
2. Auf dem Computer nach Treibersoftware suchen.
3. Aus einer Liste von Gerätetreibern auf dem Computer auswählen.
4. Von "Faulhaber MC3 WinUSB" auf "Serielles USB-Gerät" wechseln.

Im Gerätemanager wird angezeigt, welche COM-Port-Nummer zugeordnet wurde. Sie lässt sich bei Bedarf ändern.

Nach der Treiberumstellung kann der Motion Controller über USB wie ein RS232-Gerät mit den im Kommunikationshandbuch beschriebenen Diensten bedient werden.

Nach der Treiberumstellung kann der Motion Controller auch mit dem Motion Manager nur noch über den virtuellen COM-Port anstatt über USB angesprochen werden.

7.2.2 Verwendung der Moman USB-DLL

Die Motion Manager Interface-DLL MC3Usb.dll kann ebenfalls in eine eigene Anwendung eingebunden werden. Die Schnittstellenfunktionen sind in der Header-Datei **Momanusb.h** definiert:

```
void mmUsbInitLib(tdmmUsbDataCallback DataReceived);

void mmUsbDeinitLib(void);

int mmUsbOpen(int port, int channel);

void mmUsbClose(void);

void mmUsbSendData(char* data, int len);

int mmUsbReadData(char* data);

int mmUsbGetBufCount(int direction);
```

Die an **mmUsbInitLib()** übergebene Callback-Funktion wird bei eintreffenden Nachrichten aufgerufen. Anschließend müssen die empfangenen Daten über **mmUsbReadData()** eingelesen und entsprechend der Dokumentation im Kommunikationshandbuch interpretiert werden.

USB-Telegramme werden mit der Funktion **mmUsbSendData()** versendet.

7.3 CAN

Die Hersteller von CAN-Interface-Karten (z. B. HMS-IXXAT, Peak) bieten eigene API-Libraries zur Verwendung mit unterschiedlichen Programmiersprachen für ihre Treiber an.

8 Funktionsübersicht

MomanLib stellt folgende API-Funktionen zur Verfügung:

mmIntfEnumPorts

mmIntfIsPortAvailable

mmProtChangeBaudrate

mmProtChangeNodeNr

mmProtCheckCommand

mmProtCheckMotionCommand

mmProtCloseCom

mmProtCloseInterface

mmProtConnect

mmProtDecodeAnswStr

mmProtDecodeCmdStr

mmProtFindConnection

mmProtGetAbortMessage

mmProtGetCommunicationHistory

mmProtGetCommunicationSettings

mmProtGetInt64Obj

mmProtGetObj

mmProtGetSendTelegram

mmProtGetStrObj

mmProtInitInterface

mmProtLoadCommandSet

mmProtNetworkService

mmProtOpenCom

mmProtReadAnswer

mmProtRequestTrace

mmProtScanNode

mmProtSendBuffer

mmProtSendCommand

mmProtSendMotionCommand

mmProtSendTelegram

mmProtSetDataCallback

mmProtSetObj

mmProtSetObjTimeout

mmProtSetStrObj

mmProtSetTraceValuesCallback

mmProtSetupMessageFilter

mmProtSetupTrace

mmProtSupportedBaudratesList

mmProtUnconfiguredSlavesCount

mmProtWaitAnswer

9 FAULHABER Lizenzvertrag

End User Lizenzvertrag für Software der
Dr. Fritz Faulhaber GmbH & Co. KG

zwischen

(1) **Dr. Fritz Faulhaber GmbH & Co. KG**, Faulhaberstraße 1, 71101 Schönaich
- nachfolgend "FAULHABER" -

und

(2) Ihnen als Anwender
- nachfolgend "Lizenznehmer" -

Die Parteien zu (1) und (2) werden nachfolgend auch gemeinsam als die "**Parteien**" und einzeln als eine "**Partei**" bezeichnet.

VORBEMERKUNG

(A) FAULHABER konstruiert Antriebssysteme und stellt diese her. Zudem hat FAULHABER verschiedene Softwareprodukte entwickelt. Beispielsweise ermöglicht der "FAULHABER Motion Manager" (nachfolgend "**Motion Manager**") die Inbetriebnahme und Konfiguration von FAULHABER Antriebssystemen. Einzelheiten ergeben sich - soweit vorhanden - aus dem zum jeweiligen Softwareprodukt gehörigen Handbuch. Soweit nicht ausdrücklich anders geregelt, wird das Softwareprodukt dem Lizenznehmer ohne zusätzliche Vergütung als Ergänzung zu anderen von FAULHABER angebotenen Hard- und Softwareprodukten zur Verfügung gestellt.

(B) Der Lizenznehmer beabsichtigt ein oder mehrere Softwareprodukt(e) in seinem Unternehmen einzusetzen. FAULHABER ist dazu bereit, dem Lizenznehmer zu den Bedingungen dieses End User Lizenzvertrags für Software der Dr. Fritz Faulhaber GmbH & Co. KG (nachfolgend "**Vertrag**") an dem bzw. den Softwareprodukten Nutzungsrechte einzuräumen. Die Einzelheiten hierzu ergeben sich aus **§ 2**.

Dies vorausgeschickt, vereinbaren die Parteien was folgt:

§ 1
Vertragsgegenstand

(1) Gegenstand dieses Vertrags ist die Überlassung eines oder mehrerer der in Absatz (2) aufgeführten Softwareprodukte (nachfolgend "**Lizenzgegenstand**") und die Einräumung der in **§ 2** beschriebenen Nutzungsrechte durch FAULHABER an den Lizenznehmer.

(2) Die Regelungen dieses Vertrags betreffen die nachfolgend aufgeführten Kategorien von Lizenzgegenständen, inklusiv dazugehöriger Handbücher, sofern vorhanden:

a) Motion Manager mit dazugehöriger Benutzerdokumentation;

b) Programmierbibliotheken;

c) Firmware;

d) Ablaufprogramme.

(3) Nicht Gegenstand dieses Vertrags sind insbesondere folgende Leistungen:

a) Installation oder sonstige Einrichtung des Lizenzgegenstands beim Lizenznehmer;

b) etwaige individuelle Einstellungen von variablen Parametern des Lizenzgegenstands entsprechend den Anforderungen des Lizenznehmers (Customizing);

c) individuelle Programmerweiterungen für den Lizenznehmer (individuelle Modifikationen);

d) Anpassungen von Schnittstellen an die Bedürfnisse des Lizenznehmers;

e) Einweisung und Schulung der Programmbenutzer des Lizenznehmers;

f) Pflege des Lizenzgegenstands, insbesondere Lieferung neuer, zukünftiger Versionen.

§ 2
Inhalt und Umfang der Nutzungsrechte

(1) Alle Rechte am Lizenzgegenstand stehen ausschließlich FAULHABER oder ihren Lizenzgebern zu. Dem Lizenznehmer stehen am Lizenzgegenstand ausschließlich die in diesem Vertrag vereinbarten Rechte zu.

(2) Die Parteien sind sich darüber einig, dass der Lizenzgegenstand und die zugehörigen Dokumente, einschließlich zukünftiger Versionen urheberrechtlich geschützt sind und nach Maßgabe des **§ 6** Vertrauliche Informationen sowie Geschäftsgeheimnisse von FAULHABER darstellen.

(3) Soweit in Absatz (4) nicht anders geregelt, räumt FAULHABER dem Lizenznehmer die folgenden Nutzungsrechte am Lizenzgegenstand ein:

a) FAULHABER räumt dem Lizenznehmer das räumlich und zeitlich unbeschränkte, nicht ausschließliche Recht ein, den Lizenzgegenstand für die eigenen Zwecke und für die Zwecke des Kunden des Lizenznehmers bestimmungsgemäß nach Maßgabe der folgenden Absätze zu nutzen.

b) Dieses Recht umfasst die Installation des Lizenzgegenstands sowie das Laden, Anzeigen und Ablaufenlassen des installierten Lizenzgegenstands sowie das Speichern des Lizenzgegenstands im Arbeitsspeicher der Hardware, auf dem der Lizenzgegenstand installiert ist. Der Lizenznehmer ist insbesondere nicht berechtigt, den Lizenzgegenstand zu bearbeiten oder sonst zu verändern, es sei denn, dies ist in diesem Vertrag ausdrücklich gestattet.

c) Vervielfältigungen des Lizenzgegenstands sind nur insoweit zulässig, als dies für den vertragsgemäßen Gebrauch notwendig ist. Der Lizenznehmer darf vom Lizenzgegenstand Sicherungskopien nach den Regeln der Technik im notwendigen Umfang und in unveränderter Form anfertigen, und zwar insbesondere auch im Rahmen seiner normalen Sicherung der Systemumgebung.

d) Der Lizenznehmer ist zur Übertragung des Lizenzgegenstands berechtigt, wenn (i) der Lizenznehmer den Lizenzgegenstand gemeinsam mit originalen Hardwarekomponenten von FAULHABER weitergibt, (ii) die Weitergabe des Lizenzgegenstands für den Dritten unentgeltlich erfolgt, (iii) der Lizenznehmer sicherstellt, dass dem Dritten keine weitergehenden Rechte an dem Lizenzgegenstand eingeräumt werden als dem Lizenznehmer nach diesem Vertrag zustehen und (iv) dem Dritten mindestens die in Bezug auf den Lizenzgegenstand bestehenden Pflichten dieses Vertrags auferlegt werden. Als Dritte gelten auch solche Unternehmen, die i. S. d. § 15 AktG mit dem Lizenznehmer verbunden sind.

e) FAULHABER ist berechtigt, den Lizenzgegenstand ohne vorherige Ankündigung zu aktualisieren, z. B. um Fehler zu beheben oder Funktionen zu verbessern oder zu erweitern. Ersetzt die aktualisierte Version den zuvor überlassenen Lizenzgegenstand, so unterliegt diese ebenfalls den Bestimmungen dieses Vertrags.

f) Der Lizenznehmer darf den Lizenzgegenstand nur im Rahmen der bestimmungsgemäßen Verwendung und nur dann für den produktiven Betrieb nutzen, wenn der Lizenzgegenstand für den konkreten Anwendungsfall qualifiziert ist. "**Produktiver Betrieb**" bedeutet die Ansteuerung des jeweiligen durch FAULHABER hergestellten Antriebssystems durch den Lizenzgegenstand im laufenden Betrieb der Anwendung im konkreten Einsatzbereich des Lizenznehmers, allein oder in Kombination mit weiteren Komponenten eines Gesamtsystems. Die Qualifizierung für den konkreten Anwendungsfall setzt insbesondere voraus, dass entsprechende Tests in der Produktivumgebung in ausreichender Weise erfolgreich durchgeführt wurden und für den konkreten Anwendungsfall bestehende rechtliche Vorgaben und Anforderungen bei der Nutzung vom Lizenznehmer vollständig erfüllt werden (z. B. internationale Standards und Normen). Dies gilt insbesondere für den Einsatz zu medizintechnischen und militärischen Zwecken sowie in sicherheitskritischen Bereichen (z. B. im Bereich der Luft- und Raumfahrt sowie zur Steuerung kerntechnischer Anlagen).

g) Der Lizenznehmer hat gegen FAULHABER keinen Anspruch auf Herausgabe des Quellcodes oder der Quellcodedokumentation. Abweichend hiervon ist der Quellcode des Lizenzgegenstands Bestandteil der Nutzungseinräumung, soweit dies in diesem Vertrag (insbesondere in Abs. (4) unten) ausdrücklich bestimmt wird.

h) Soweit der dem Lizenznehmer von FAULHABER überlassene Lizenzgegenstand Open-Source Software oder Software enthält, für die FAULHABER nur ein abgeleitetes Nutzungsrecht besitzt (nachfolgend "**Drittsoftware**"), gelten zusätzlich und vorrangig die Nutzungsregelungen, denen diese Drittsoftware unterliegt. Die innerhalb eines Lizenzgegenstands jeweils verwendete Drittsoftware, die auf die Drittsoftware anwendbaren Lizenzbedingung(en) sowie eventuell vorhandene Urhebervermerke sind jeweils im dazugehörigen Handbuch genannt oder werden dem Lizenznehmer mit der Auslieferung des Lizenzgegenstands in einer separaten Datei zum Download zur Verfügung gestellt. Der Lizenznehmer ist verpflichtet, die jeweiligen Lizenzbedingungen einzuhalten. Im Fall der Verletzung dieser Lizenzbedingungen durch den Lizenznehmer sind neben FAULHABER auch die Lizenzgeber berechtigt, die daraus entstehenden Ansprüche und Rechte im eigenen Namen geltend zu machen.

(4) Die Nutzungsrechte gemäß Absatz (3) werden für die nachfolgend genannten Lizenzgegenstände wie folgt ergänzt bzw. modifiziert:

a) Motion Manager

aa) Die bestimmungsgemäße Verwendung des Motion Managers ergibt sich - soweit vorhanden - aus der jeweils aktuellen Version des zugehörigen Handbuchs, das auf der Webseite von FAULHABER zur Verfügung gestellt wird.

bb) Der Motion Manager darf nur verwendet werden, wenn der Lizenznehmer sicherstellt, dass bei dessen Verwendung keine Verletzung oder Schädigung der Gesundheit und keine Gefahr materieller Schäden für Sachwerte (z. B. Anlagen) möglich ist.

cc) Der Lizenznehmer darf den Motion Manager nicht im produktiven Betrieb verwenden. Klarstellend halten die Parteien fest, dass eine solche Verwendung keine bestimmungsgemäße Nutzung des Motion Managers darstellt. Dasselbe gilt für die Verwendung zur Ansteuerung von Antriebssystemen, die nicht durch FAULHABER hergestellt wurden, sowie die Verwendung zur Ansteuerung von Antriebssystemen, die zwar durch FAULHABER hergestellt wurden, die aber nicht in der Programmbeschreibung aufgeführt sind. Abweichend hiervon gilt, dass im Motion Manager enthaltene Ablaufprogramme zur Verwendung im produktiven Betrieb angepasst und genutzt werden dürfen, sofern diese gemäß Absatz (3) f) in der Anwendung qualifiziert wurden.

dd) Das Dekompilieren sowie sonstige Arten des Reverse Engineering sind grundsätzlich unzulässig. Hiervon ausgenommen ist das Recht des Lizenznehmers, das Funktionieren des Motion Managers zu beobachten, zu untersuchen oder zu testen, um die einem Programmelement zugrundeliegenden Ideen und Grundsätze zu ermitteln, wenn dies durch Handlungen zum Laden, Anzeigen, Ablaufen, Übertragen oder Speichern des Programms geschieht, zu denen er nach Maßgabe dieses Vertrags berechtigt ist (§ 69d Abs. 3 UrhG). Zudem ist der Lizenznehmer abweichend von Satz 1 zur Dekompilierung zu Zwecken der Herstellung eines interoperablen Programms ausschließlich unter den Bedingungen des § 69e Abs. 1 und in den Schranken des § 69e Abs. 2 UrhG berechtigt. Die vorstehenden Rechte bestehen nur, wenn der Lizenznehmer vor jeder derartigen Handlung die von ihm benötigten Informationen bei FAULHABER angefragt und nicht innerhalb angemessener Zeit die erforderlichen Informationen erhalten hat. Im Rahmen seiner Anfrage hat der Lizenznehmer FAULHABER sämtliche zur Beurteilung der Anfrage erforderlichen Informationen bereitzustellen.

ee) Jede weitergehende Nutzung des Motion Managers, insbesondere die Einräumung von Unterlizenzen, bedarf der vorherigen ausdrücklichen und schriftlichen Zustimmung von FAULHABER. Dies gilt nicht bei einem Verkauf der Antriebssysteme, soweit deren ordnungsgemäße Nutzung den Einsatz des Motion Managers erfordert.

ff) Die Nutzung des Motion Managers ist ausschließlich in Verbindung mit originalen Hardwarekomponenten von FAULHABER gestattet. Die Nutzung für Fremdhardware ist untersagt.

b) Programmierbibliotheken

aa) Dem Lizenznehmer wird das Recht eingeräumt, Quellcode-Dateien der Programmierbibliotheken zu bearbeiten und diese Bearbeitungen an Dritte zu übertragen. Die Bearbeitung von Objektcode-Dateien der Programmierbibliotheken ist dagegen untersagt.

bb) Die Nutzung der Programmierbibliotheken ist ausschließlich in Verbindung mit originalen Hardwarekomponenten von FAULHABER gestattet. Die Nutzung für Fremdhardware ist untersagt.

cc) FAULHABER stellt dem Lizenznehmer bei Bedarf und nach eigenem Ermessen Handbücher für die Programmierbibliotheken auf der Webseite von FAULHABER zum Download zur Verfügung. Ein Anspruch des Lizenznehmers auf Bereitstellung eines Handbuchs besteht nicht. Soweit eine bestimmungsgemäße Verwendung der Programmierbibliotheken festgelegt ist, ergibt sich diese - soweit vorhanden - aus der jeweils aktuellen Version des zugehörigen Handbuchs.

dd) Die Programmierbibliotheken dürfen nur verwendet werden, wenn der Lizenznehmer sicherstellt, dass bei ihrer Verwendung keine Verletzung oder Schädigung der Gesundheit und keine Gefahr materieller Schäden für Sachwerte (z. B. Anlagen) möglich ist.

c) Firmware

aa) Die bestimmungsgemäße Verwendung der Firmware ergibt sich aus der jeweils aktuellen Version des zugehörigen Handbuchs, das auf der Webseite von FAULHABER zur Verfügung gestellt wird.

bb) Die Firmware darf nur verwendet werden, wenn der Lizenznehmer sicherstellt, dass bei ihrer Verwendung keine schweren Verletzungen oder erhebliche Schäden für Sachwerte (z. B. Anlagen) möglich ist.

cc) Die Nutzung der Firmware ist ausschließlich in Verbindung mit originalen Hardwarekomponenten von FAULHABER gestattet. Das Recht zur Nutzung der Firmware für Hardware eines Dritten besteht nur nach vorheriger schriftlicher Zustimmung durch FAULHABER.

dd) Das Dekompilieren sowie sonstige Arten des Reverse Engineering sind grundsätzlich unzulässig. Hiervon ausgenommen ist das Recht des Lizenznehmers, das Funktionieren der Firmware zu beobachten, zu untersuchen oder zu testen, um die einem Programmelement zugrundeliegenden Ideen und Grundsätze zu ermitteln, wenn dies durch Handlungen zum Laden, Anzeigen, Ablaufen, Übertragen oder Speichern des Programms geschieht, zu denen er nach Maßgabe dieses Vertrages berechtigt ist (§ 69d Abs. 3 UrhG). Zudem ist der Lizenznehmer abweichend von Satz 1 zur Dekompilierung zu Zwecken der Herstellung eines interoperablen Programms ausschließlich unter den Bedingungen des § 69e Abs. 1 und in den Schranken des § 69e Abs. 2 UrhG berechtigt. Die vorstehenden Rechte bestehen nur, wenn der Lizenznehmer vor jeder derartigen Handlung die von ihm benötigten Informationen bei FAULHABER angefragt und nicht innerhalb angemessener Zeit die erforderlichen Informationen erhalten hat. Im Rahmen seiner Anfrage hat der Lizenznehmer FAULHABER sämtliche zur Beurteilung der Anfrage erforderlichen Informationen bereitzustellen.

d) Ablaufprogramme

aa) Ablaufprogramme sind Programme, die auf spezifischer FAULHABER Controller Hardware ausführbar sind.

bb) Dem Lizenznehmer wird das Recht eingeräumt, Ablaufprogramme zu bearbeiten und diese Bearbeitungen an Dritte zu übertragen, sofern diese in Quellcodeform ausgeliefert wurden.

cc) Die Nutzung der Ablaufprogramme ist ausschließlich in Verbindung mit originalen Hardwarekomponenten von FAULHABER gestattet. Das Recht zur Nutzung der Ablaufprogramme auf der Hardware eines Dritten besteht nur nach vorheriger schriftlicher Zustimmung durch FAULHABER.

§ 3
Lieferung

(1) Der Lizenzgegenstand wird in der zum Zeitpunkt der Auslieferung vorhandenen Form ("as is") geliefert.

(2) Die Lieferung des Lizenzgegenstands erfolgt in digitaler Form durch Bereitstellung zum Download auf der Webseite von FAULHABER oder individuell per E-Mail. FAULHABER ist nicht verpflichtet, den Lizenzgegenstand auf physischen Datenträgern bereitzustellen.

(3) FAULHABER prüft den Lizenzgegenstand vor Bereitstellung mit einem zum Zeitpunkt der jeweiligen Bereitstellung aktuellen Virenscanner auf etwaig vorhandene Schadsoftware. Weitergehende Pflichten von FAULHABER in Bezug auf die Freiheit von Schadsoftware bestehen nicht.

§ 4
Pflichten des Lizenznehmers

(1) Der Lizenznehmer ist verpflichtet, für eine ausreichende technische Betriebs- und Systemumgebung und für den ordnungsgemäßen Betrieb des Lizenzgegenstands zu sorgen. Die Einrichtung der Betriebs- und Systemumgebung für den Lizenzgegenstand liegt allein in der Verantwortung des Lizenznehmers.

(2) Wenn der produktive Betrieb des Lizenzgegenstands nach diesem Vertrag gestattet ist, hat der Lizenznehmer sicherzustellen, dass die Anforderungen gemäß A.I. § 2(3)f) vor dem produktiven Betrieb des Lizenzgegenstands vollständig erfüllt sind. § 377 HGB bleibt unberührt.

(3) Für die Installation und Implementierung des Lizenzgegenstands auf den Systemen des Lizenznehmers ist dieser allein verantwortlich.

(4) Der Lizenznehmer hat alle erforderlichen und zumutbaren Maßnahmen zu ergreifen, um Schäden durch den Lizenzgegenstand zu verhindern oder zu begrenzen. Insbesondere hat der Lizenznehmer die aktuellen Schutzmechanismen zur Abwehr von Schadsoftware einzusetzen.

(5) Dem Lizenznehmer ist bekannt, dass für die etwaig erforderlichen Kommunikationsschnittstellen (insbesondere beim Motion Manager und Programmierbibliotheken) unter Umständen separate Treiber der Adapter-Hersteller erforderlich sind, die nicht von FAULHABER bereitgestellt werden. Kommunikationsschnittstellen sind Schnittstellen zum Datenaustausch zwischen PC und Controller z.B. über CAN, RS232, USB oder EtherCAT. Der Lizenznehmer ist verpflichtet, erforderliche Treiber zur Nutzung dieser Kommunikationsschnittstellen eigenständig zu beschaffen, zu lizenzieren und zu installieren.

(6) Dem Lizenznehmer ist es untersagt, etwaige Urheberrechtsinformationen aus dem Lizenzgegenstand zu entfernen oder abzuändern.

§ 5
Haftung von FAULHABER

(1) Soweit der Lizenzgegenstand unentgeltlich überlassen wird, geltend folgende Haftungsregelungen:

a) FAULHABER haftet für Sach- und Rechtsmängel nur dann, wenn FAULHABER einen solchen Mangel arglistig verschweigt.

b) FAULHABER haftet bei Vorsatz, grober Fahrlässigkeit und Ansprüchen nach dem Produkthaftungsgesetz, arglistigem Verschweigen eines Mangels, Garantieansprüchen sowie bei einer Verletzung des Lebens, des Körpers oder der Gesundheit nach den gesetzlichen Vorschriften. Im Übrigen ist die Haftung von FAULHABER für Schadens- und Aufwendungsersatzansprüche - gleich aus welchem Grund - ausgeschlossen.

c) Soweit ein Verlust oder eine Zerstörung von Daten beim Lizenznehmer durch grob fahrlässige oder vorsätzliche Verletzung vertraglicher oder gesetzlicher Pflichten verursacht wurde, haftet FAULHABER nur bis zur Höhe des typischen Wiederherstellungsaufwands, der trotz regelmäßiger, dem Stand der Technik entsprechender Datensicherung entsteht.

(2) Für den Fall, dass Firmware zum Aufspielen auf Hardwarekomponenten von FAULHABER bereitgestellt wird, gelten abweichend von Absatz (1) die auf die jeweilige Hardwarekomponente anwendbaren Haftungsregelungen.

§ 6
Vertraulichkeit

(1) Die Parteien verpflichten sich, alle im Rahmen der Vertragsanbahnung und -durchführung erlangten Kenntnisse von als vertraulich gekennzeichneten oder ihrer Natur nach vertraulichen Informationen ("**Vertrauliche Informationen**") der jeweils anderen Partei zeitlich unbegrenzt vertraulich zu behandeln und nur für Zwecke der Durchführung dieses Vertrages zu verwenden. Zu den Vertraulichen Informationen von FAULHABER gehört auch der Lizenzgegenstand. Abweichend hiervon gelten - soweit nicht anders vereinbart - die von FAULHABER auf der Webseite von FAULHABER öffentlich zum Download zur Verfügung gestellten Lizenzgegenstände nicht als Vertrauliche Informationen.

(2) Der Lizenznehmer wird den Lizenzgegenstand Mitarbeitern und sonstigen Dritten nur zugänglich machen, soweit dies zur Ausübung der ihm eingeräumten Nutzungsbefugnisse erforderlich ist. Er wird alle Personen, denen er Zugang zum Lizenzgegenstand gewährt über die daran bestehenden Rechte von FAULHABER und die Geheimhaltungspflicht belehren und diese Personen schriftlich in gleichem Maße wie in diesem § 6 zur Geheimhaltung verpflichten, soweit die betreffenden Personen nicht aus anderen Rechtsgründen zur Geheimhaltung mindestens in vorstehendem Umfang verpflichtet sind.

(3) Die Verpflichtungen zur Geheimhaltung nach den vorstehenden Absätzen gelten nicht für Vertrauliche Informationen, die (i) zur Zeit ihrer Übermittlung durch die Partei bereits offenkundig oder der anderen Partei bekannt waren; (ii) nach ihrer Übermittlung durch die Partei ohne Verschulden der anderen Partei offenkundig geworden sind; (iii) nach ihrer Übermittlung durch die Partei der anderen Partei von dritter Seite auf nicht rechtswidrige Weise und ohne Einschränkung in Bezug auf Geheimhaltung oder Verwertung zugänglich gemacht worden sind; und/oder (iv) die von einer Partei eigenständig, ohne Nutzung der Vertraulichen Informationen oder der Betriebsgeheimnisse der anderen Partei, entwickelt worden sind. Die Verpflichtungen gelten weiterhin nicht, soweit die Vertraulichen Informationen gemäß Gesetz, und zwar insbesondere aufgrund behördlicher Verfügung oder gerichtlicher Entscheidung veröffentlicht werden müssen; insoweit wird die veröffentlichende Partei die andere Partei hierüber unverzüglich informieren und sie in der Abwehr derartiger Verfügungen bzw. Entscheidungen unterstützen.

§ 7
Schlussbestimmungen

(1) Änderungen oder Ergänzungen dieses Vertrags bedürfen der Schriftform. Genügen sie dieser nicht, so sind sie nichtig. Dies gilt auch für Änderungen dieser Schriftformklausel.

(2) Dieser Vertrag unterliegt dem Recht der Bundesrepublik Deutschland. Die Geltung des UN-Kaufrechts (CISG United Nations Convention on Contracts for International Sale of Goods vom 11.04.1980) ist ausgeschlossen.

(3) Ausschließlicher Gerichtsstand ist Stuttgart, wenn der Lizenznehmer Kaufmann im Sinne des Handelsgesetzbuches, juristische Person des öffentlichen Rechts oder öffentlich-rechtliches Sondervermögen ist oder bei Klageerhebung keinen Sitz oder gewöhnlichen Aufenthaltsort in der Bundesrepublik Deutschland hat.

(4) Sollte eine Bestimmung dieses Vertrags unwirksam sein oder werden, so bleiben alle übrigen Bestimmungen hiervon unberührt. An die Stelle von nicht einbezogenen oder unwirksamen Bestimmungen tritt das Gesetzesrecht (§ 306 Abs. 2 BGB). Im Übrigen werden die Parteien anstelle der nichtigen oder unwirksamen Bestimmung eine wirksame Regelung treffen, die ihr wirtschaftlich möglichst nahekommt, soweit keine ergänzende Vertragsauslegung vorrangig oder möglich ist.

DR. FRITZ FAULHABER GMBH & CO. KG
Antriebssysteme

Faulhaberstraße 1
71101 Schönaich • Germany
Tel. +49(0)7031/638-0
Fax +49(0)7031/638-100
info@faulhaber.de
www.faulhaber.com

7000.00064, 2. Auflage, 8.10.2022