STUDIENARBEIT

im Rahmen des Softwaresanierungprojekts
mit dem Thema:

Beschreibung einer Schnittstelle zur Motorenansteuerung: Das C-Interface des RTK-Steuerprogramms

von
Sebastian Freund
Derrick Hepp

Institut für Informatik
Math.-Naturwiss. Fakultät II
Lehrstuhl für Softwaretechnik
Rudower Chaussee 25
Humboldt-Universität zu Berlin

Kapitel 1

Schnittstellenbeschreibung der Motorenansteuerung

Vom ursprünglichen Entwickler des Programms ist die Hardwareansteuerung, im Speziellen der Zugriff auf die Motoren, von den übrigen Programmteilen gut abgekapselt worden. Der Programmierer hat versucht, alle für die Motoren relevanten Funktionen in einer Dynamic Link Library (DLL)[1], der Bibliotheksdatei `motors.dll`, zusammenzufassen.

Für jede Art der Motorenansteuerung, die grundsätzlich von der verwendeten Hardware-Motorsteuerkarte abhängt, ist im Programm eine spezielle Klasse in C++ implementiert. Um die Motorenhardware auch softwaretechnisch ohne Kenntnisse der Objektorientierung nutzen zu können, hat der Entwickler ein Interface in der Programmiersprache C entworfen. Damit ist es möglich, die Motoren mittels dieser Bibliothek außerhalb des Röntgentopographie-Steuerprogramms anzusteuern.

Dieses C-Interface stellt eine Abstraktionsschicht (High-Level-Layer) der Motoransteuerung dar. Der Benutzer benötigt hierbei keine Informationen über die Art des Hardwarezugriffs, sondern kann die Motoren über die ihnen zugeordnete Achse oder ihre Identifikationsnummer ansprechen und die gewünschte Funktion ausführen lassen.

Im Folgenden werden die vom Entwickler bereitgestellten C-Funktionen und ihre Anwendung beschrieben.

[1] Bibliothek von Funktionen, die erst zur Laufzeit dynamisch vom Hauptprogramm geladen wird

Zuerst muß zwischen zwei Arten von Funktionen unterschieden werden:

a) ml*FunctionName* - Diese Funktionen benutzen den Motor mit der übergebenen Motornummer. „ml" steht hierbei für Motorliste, das heißt, es wird auf Funktionen und Membervariablen der Klasse TMList zugegriffen.

b) m*FunctionName* - Diese Funktionen benutzen den zur Zeit aktivierten Motor. „m" steht hierbei für Motor, das bedeutet, daß Funktionen und Membervariablen der von TMotor abgeleiteten Klassen benutzt werden.

1.1 Interface der Motorlisten-Funktionen

Der Großteil der Motorlisten-Funktionen erwartet als Übergabeparameter eine Motor-Identifikationsnummer *mid*, die festlegt, für welchen Antrieb die jeweilige Funktion ausgeführt werden soll.

mlInitializeMotorsDLL

Syntax:	`BOOL WINAPI mlInitializeMotorsDLL( void )`
Beschreibung:	Diese Funktion führt die Initialisierung der Bibliothek `motors.dll` aus. Die Konfigurationsdatei wird nach Einträgen der verschiedenen Motorobjekte (z.B. Objekte der Klassen TMotor, TC812ISA, TC832) durchsucht, die dann erzeugt und der Reihenfolge nach in die Motorenliste (Klasse TMList) aufgenommen werden. Die Membervariablen der Motoren werden mit Werten aus der INI-Datei belegt. Es wird versucht, die Motoren über die Motorsteuerungshardware anzusprechen und zu initialisieren.
Parameter:	-
Rückgabewert:	`TRUE` = Initialisierung erfolgreich `FALSE` = Initialisierung mindestens eines Motors schlug fehl
Siehe auch:	`mlIsServerOK`

mlSetAxis

Syntax:	`BOOL WINAPI mlSetAxis( int mid )`
Beschreibung:	Festlegung der aktiven Achse. Damit wird der Antrieb mit der angegebenen Motor-ID als aktiver Motor eingestellt, indem die TMList-Membervariable *nActiveDrive* mit der übergebenen Motor-Identifikationsnummer *mid* belegt wird. Der Antrieb der so aktivierten Achse kann danach direkt über die m*FunctionName*-Funktionen angesprochen werden.
Parameter:	`int` *mid* - Identifikationsnummer des Motors der speziellen Achse
Rückgabewert:	`TRUE` = Auswahl war erfolgreich `FALSE` = Auswahl schlug fehl, weil die Achse nicht existiert
Siehe auch:	`mlGetAxis`, `mlGetIdByName`

mlGetAxis

Syntax:	`int WINAPI mlGetAxis( void )`
Beschreibung:	Ermittelt die Identifikationsnummer der derzeit aktivierten Achse.
Parameter:	-
Rückgabewert:	Die Membervariable *nActiveDrive* der Klasse TMList, die die Identifikationsnummer des Motors der aktiven Achse speichert, wird zurückgeliefert.
Siehe auch:	`mlSetAxis`, `mlGetIdByName`

mlGetIdByName

Syntax:	`int WINAPI mlGetIdByName( TAxisType axis)`
Beschreibung:	Gibt die Motornummer (ID) des entsprechenden Achsenmotors zurück.
Parameter:	`TAxisType` *axistype* - zulässige Parameter sind X, Y, Z, Omega, Theta, Phi, Psi, Encoder, Monochromator, Absorber, Collimator, DF, DC, Tilt, Rotation. Die Struktur `TAxisType` ist in der Datei `comhead.h` deklariert.
Rückgabewert:	≥ 0 = Identifikationsnummer (ID) des zur übergebenen Achse gehörigen Motors -1 = Fehler, d.h. die übergebene Achse wurde noch nicht initialisiert
Siehe auch:	`mlSetAxis`, `mlGetAxis`, `mGetAxisName`

mlGetDistance

Syntax:	`BOOL WINAPI mlGetDistance(int mid,double &position)`
Beschreibung:	Ermittelt für den Antrieb mit der angegebenen Motor-ID die aktuelle Stellung im Winkelmaß (Grad etc.). Die Position wird von der Hardware ausgelesen. Falls sich der Motor in Ruhe befindet und kein Übertragungsfehler aufgetreten ist, wird die Motorposition in der Motorklassenvariablen *dAngle* gespeichert.
Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse `double &` *position* - Rückgabeparameter für die ausgelesene Position
Rückgabewert:	`TRUE` = Die Position konnte ermittelt werden, wurde in der Motor-Membervariablen *dAngle* abgelegt und wird im `double&` Parameter als Winkel zurückgeliefert. `FALSE` = Fehlerfall: Motor war in Fahrt oder die Position konnte nicht ausgelesen werden

Siehe auch: `mlGetValue`, `mGetDistance`, `mGetValue`

mlGetValue

Syntax: `double WINAPI mlGetValue(int mid,TValueType vtype)`

Beschreibung: Abhängig vom Wert des Parameters *vtype* wird der dazugehörige Wert der Motorgröße derjenigen Achse zurückgeliefert, die durch *mid* bestimmt wird. Der Typ TValueType (deklariert in `comhead.h`) kann folgende Werte annehmen: *Distance* (Winkelstellung des Motors), *MinDistance*, *MaxDistance*, *Speed*, *Width* (Winkelschrittweite). Die Funktion wertet nur die Membervariablen der Motoren aus. Es findet kein Zugriff auf die Hardware statt.

Parameter: `int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
`TValueType` *vtype* - Parameter gibt an, welcher Motorwert zurückgeliefert werden soll.

Rückgabewert: `double` Wert der durch den `TValueType` Parameter bestimmten Membervariablen des Antriebs, wobei folgende Zuordnung stattfindet: *Distance* = *dAngle*, *MinDistance* = *dAngleMin*, *MaxDistance* = *dAngleMax*, *Speed* = *dSpeed* unter Berücksichtigung von *dwVelocity* bzw. *Width* = *dAngleWidth*

Siehe auch: `mlGetDistance`, `mGetValue`, `mGetDistance`, `mSetValue`

mlMoveToDistance

Syntax: `BOOL WINAPI mlMoveToDistance(int mid,double distance)`

Beschreibung: Startet die Bewegung des Motors, der durch *mid* vorgegeben wird, an die durch *distance* angegebene Motorstellung im Winkelmaß.

Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse `double` *distance* - Winkelposition, an die der Motor bewegt werden soll
Rückgabewert:	`FALSE` = Fehlerfall: `motors.dll` konnte nicht korrekt initialisiert werden, d.h. die Konfigurationsdatei wurde nicht erfolgreich eingelesen. `TRUE` = Da keine Fehlerkontrolle beim Aufruf der Motorlisten-Funktionen vorgenommen wird, ist die Funktion in jedem anderen Fall (auch bei ungültiger Winkelangabe im Parameter *distance*) erfolgreich.
Siehe auch:	`mMoveToDistance`, `mMoveByDistance`, `mlGetDistance`

mlIsMoveFinish

Syntax:	`BOOL WINAPI mlIsMoveFinish( int mid )`
Beschreibung:	Stellt fest, ob der Motor mit der angegebenen Motor-ID die anzufahrende Winkelposition erreicht hat.
Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
Rückgabewert:	`TRUE` = Antrieb hat die vorgegebene Stellung angefahren und befindet sich in Ruhe. `FALSE` = Motor ist in Bewegung oder an einer Endlage (Deathband) zum Stehen gekommen.
Siehe auch:	`mIsMoveFinish`, `mlMoveToDistance`

mlGetOffset

Syntax:	`double WINAPI mlGetOffset( int mid )`
Beschreibung:	Gibt das Offset für die Relative Null des angeforderten Antriebs zurück. Der Wert ist in der Motor-Membervariablen *dAngleBias* abgelegt.

Parameter:	`int` *mid* - Identifikationsnummer des Motors einer speziellen Achse
Rückgabewert:	`double` Wert der Membervariablen *dAngleBias* des angegebenen Motors. Ist der Wert $\neq 0$, dann wurde die Relative Null gesetzt, und alle Operationen berücksichtigen das Offset.
Siehe auch:	`mSetRelativeZero`, `mIsDistanceRelative`

mlParsingAxis

Syntax:	`TAxisType WINAPI mlParsingAxis( LPSTR axisname )`
Beschreibung:	Umwandlung eines Strings in einen TAxisType (deklariert in `comhead.h`). Dadurch ist es möglich, für eine Achse mehrere Bezeichnungen zu wählen. Z.B. werden die Zeichenketten *AzimutalRotation*, *AZ*, *Rotation* und *Azimute* demselben Achsentyp *Rotation* zugeordnet.
Parameter:	`LPSTR` *axisname* - Zeichenkette des Achsennamens
Rückgabewert:	`TAxisType` = Typ der zugeordneten Motorachse. Folgende Werte sind möglich: X, Y, Z, Omega, Theta, Phi, Psi, Encoder, Absorber, Tilt, Collimator, Rotation, Monochromator, DC, DF.
Siehe auch:	`mGetAxisName`, `mlGetIdByName`, `mlIsAxisValid`

mlIsAxisValid

Syntax:	`BOOL WINAPI mlIsAxisValid( TAxisType axis )`
Beschreibung:	Ermittelt, ob die bestimmte Achse initialisiert wurde, d.h., ein der Motorachse zugeordneter Motor wurde im System gefunden und entsprechend den Einträgen in der Konfigurationsdatei eingerichtet.

Parameter:	`TAxisType` *axis* - Typ der geforderten Achse. Die Enumeration TAxisType ist in `comhead.h` deklariert. Folgende Werte sind möglich: X, Y, Z, Omega, Theta, Phi, Psi, Encoder, Absorber, Tilt, Collimator, Rotation, Monochromator, DC, DF.
Rückgabewert:	`TRUE` = Die angegebene Motorachse ist bereits erfolgreich initialisiert worden. `FALSE` = In der Konfigurationsdatei existiert kein Eintrag für den geforderten Antrieb, oder der Motorachsentyp wurde falsch übergegeben.
Siehe auch:	-

mlIsServerOK

Syntax:	`BOOL WINAPI mlIsServerOK( void )`
Beschreibung:	Stellt fest, ob die `motors.dll` erfolgreich geladen und die Antriebe richtig konfiguriert wurden. Dazu wird der Wert der statischen Variablen *bModulLoaded* zurückgeliefert, der bei erfolgreicher Ausführung der Funktion `mlInitializeMotorsDLL` gesetzt wird.
Parameter:	-
Rückgabewert:	`TRUE` = Die Bibliothek `motors.dll` wurde ohne Probleme eingebunden und alle Antriebe wurden korrekt konfiguriert. `FALSE` = Die Funktion `mlInitializeMotorsDLL` schlug fehl. Der Wert der Variablen *bModulLoaded* wurde nicht auf `TRUE` gesetzt. D.h. die Bibliothek `motors.dll` konnte nicht initialisiert werden.
Siehe auch:	`mlInitializeMotorsDLL`

mlGetAxisNumber

Syntax:	`int WINAPI mlGetAxisNumber( void )`
Beschreibung:	Ermittelt die Anzahl der verfügbaren Antriebsachsen. Dazu wird die Membervariable *nLastDrive* der Klasse TMList ausgewertet.
Parameter:	-
Rückgabewert:	`int` Wert der privaten TMList-Klassenvariablen *nLastDrive* + 1, entspricht Anzahl der Antriebe, die dem Steuerprogramm bekannt sind
Siehe auch:	-

mlSaveModuleSettings

Syntax:	`void WINAPI mlSaveModuleSettings( void )`
Beschreibung:	Speichert für alle im System vorhandenen Antriebe die aktuellen Motorparameter (z.B. Velocity, PositionWidth, AngleMin, AngleWidth, Acceleration, DynamicGain) in der Konfigurationsdatei unter dem zum jeweiligen Motor gehörigen Abschnitt ab. Dazu werden die `SaveSettings`-Methoden der Motorklassen aufgerufen. Außerdem wird für die einzelnen Antriebe der INI-Wert `RestartPossible` auf `TRUE` gesetzt, um ein ordnungsgemäßes Programmabschluß zu signalisieren. Die Motoren werden danach gestoppt.
Parameter:	-
Rückgabewert:	-
Siehe auch:	-

mlSetAngleDefault

Syntax:	`void WINAPI mlSetAngleDefault( void )`
Beschreibung:	Setzt die Voreinstellungen für das Winkeloffset der Relativen Null und die Softwareschranken im Winkelmaß. Dazu werden die Membervariablen *dAngleBias*, *dAngleMin* und *dAngleMax* der einzelnen im System vorhandenen Antriebe auf ihre Defaultwerte zurückgestellt. Für *dAngleMin* und *dAngleMax* werden die Membervariablen *lPositionMin* und *lPositionMax*, die die zulässigen Positionsbereiche in Encoderschritten darstellen, ausgewertet; *dAngleBias* wird auf 0.0 gesetzt.
Parameter:	-
Rückgabewert:	-
Siehe auch:	`mSetAngleDefault`

mlGetVersion

Syntax:	`LPCSTR WINAPI mlGetVersion( void )`
Beschreibung:	Rückgabe der Versionsnummer der `m_layer`-Schicht. Dazu wird der Wert der statischen Variablen *mlVersion* zurückgeliefert.
Parameter:	-
Rückgabewert:	`LPCSTR` Wert der statischen Variablen *mlVersion*, der die Versionsnummer der Motorsteuerungsschicht inklusive Datum der Kompilierung enthält
Siehe auch:	-

mlGetInstance

Syntax:	`HINSTANCE WINAPI mlGetInstance( void )`
Beschreibung:	Gibt das Instanz-Handle des Moduls `motors.dll` zurück.
Parameter:	-
Rückgabewert:	`HINSTANCE` Wert der globalen Variable *hModuleInstance*, die in der `LibMain`-Funktion der `motors.dll` mit dem Instanz-Handle für die DLL belegt wurde, wird zurückgegeben.
Siehe auch:	-

1.1.1 Dialoge zur Motorsteuerung

Vom Entwickler sind auch Dialoge zur Steuerung der Motoren in das Interface aufgenommen worden, um anderen Programmierern die Möglichkeit zu geben, auf komplexe Dialoge ohne großen Aufwand über Aufrufe aus der Dynamischen Bibliothek `motors.dll` zurückgreifen zu können, um beispielsweise innerhalb eines eigenen Programms einen Referenzpunktlauf (→ `mlInquireReferencePointDlg`) durchzuführen oder die Motoren direkt ansteuern zu können (→ `mlPositionControlDlg`).

Diese Herangehensweise ist unvorteilhaft im Hinblick auf die Trennung von Oberfläche, Hauptprogramm und Hardwareansteuerung, ist aber der Intention des Steuerprogrammentwicklers geschuldet.

mlInquireReferencePointDlg

Syntax:	`void WINAPI mlInquireReferencePointDlg( int task)`
Beschreibung:	Startet den „Grundstellung anfahren"-Dialog. Mit dieser Funktion wird ein Dialog aufgerufen, mit dem für alle Motoren des Systems ein Referenzpunktlauf durchgeführt werden kann.

Weiterhin ist es möglich, den Nullpunkt in Relation zum Referenzpunkt neu zu bestimmen und auch den absoluten Nullpunkt zu setzen. Im Steuerprogramm wird diese Funktion über das Hauptmenü (*Einstellungen / Motoren / Grundstellung*) aufgerufen.

Parameter: `int` *task* - Übergabe einer Tasknummer, die festlegt, welche Voreinstellungen der Dialog anzeigen soll. Wird *task* = 99 übergeben, wird veranlaßt, daß alle Antriebe für den Referenzpunktlauf ausgewählt werden.

---

mlOptimizingDlg

Syntax: `void WINAPI mlOptimizingDlg ( void )`

Beschreibung: In den meisten Fällen sind die Steuerparameter nur im Rahmen eines konkreten Versuchsaufbaus richtig bestimmbar. Da den meisten Antrieben zudem auch ein mechanisches Spiel anhaftet, sind die Parameter nur unter Beachtung der realen Anfahrcharakteristik optimal einstellbar. Zum Optimieren wird mit dem Dialog „manuelle Justage" der zu optimierende Antrieb ausgewählt. Die Funktion `mlOptimizingDlg` startet den „DC-Controller-Parameter"-Dialog für den aktuellen Antrieb. Zuvor wird eine `cm_CallExecuteScan`-Nachricht an die Anwendung gesendet, um das Scan-Fenster zu aktivieren. Der durchgeführte Scan tastet die Motorbewegung 150 Mal ab. Das Ergebnis dieses Scans wird dann im Scanfenster visualisiert. Ziel der Optimierung ist es, daß eine Endposition gleichmäßig und ohne Schwingungen angefahren wird.
Die änderbaren Parameter für Motoren der Steuerkarte C-812 sind die Werte der Membervariablen *dwMaxVelocity* (maximale Geschwindigkeit), *dwAcceleration* (Beschleunigung), *wStaticGain* (statische Verstärkung), *wDynamicGain* (dynamische Verstärkung), *wTorque* (Beschränkung des maximalen Motorstroms), *wPositionWidth* (Schrittweite zum Messen des Anfahrverhaltens).

Bei den Motoren an der Steuerkarte C-832 sind die Bezeichnungen für statische und dynamische Verstärkung anders gewählt: *wKP*, *wKD*. Außerdem läßt sich über *wKI* die Integralverstärkung und durch *wKL* das Integrallimit festlegen.
Der Benutzer hat die Möglichkeit, einen CheckScan durchzuführen, bei dem das Anfahrverhalten des Motors überprüft wird. Dazu ruft das Programm die Interfacefunktion `mStartMoveScan` auf.

Parameter: -

---

mlPositionControlDlg

Syntax: `void WINAPI mlPositionControlDlg ( void )`

Beschreibung: Startet den Dialog zur Positionsansteuerung für die gesamten Antriebe. In diesem Dialog kann der Antrieb in Encoderschritten[2] angesteuert werden. Der Dialog wird im Steuerprogramm über das Hauptmenü (*Einstellungen / Antriebe / Direkte Steuerung*) aufgerufen.

Parameter: -

---

mlSetParametersDlg

Syntax: `void WINAPI mlSetParametersDlg ( void )`

Beschreibung: Startet den „Motor-Parameter"-Dialog.Dieser dient der Einstellung der Softwareschranken und der Schrittweite der einzelnen Antriebe. Die Parameter können jeweils in Encoder- oder Winkeleinheiten verändert werden.
Der Dialog wird im Steuerprogramm über das Haupt-

[2]Ein Encoderschritt ist die kleinste Einheit, mit der die Motorsteuerkarten die Schrittmotoren steuern können. Die Umrechnung von Winkelangaben in diese Einheit wird vom Steuerprogramm in der Motor-Klassenfunktion `TMotor::Translate` realisiert.

menü (*Einstellungen / Antriebe / Parameter*) aufgerufen.

Parameter: -

1.2 Interface der Motor-Funktionen

Im Unterschied zu den Motorlisten-Funktionen beziehen sich die Funktionen des Interface der Motoren auf den aktuell eingestellten Antrieb, der in der Motorlisten-Membervariable *nActiveDrive* gespeichert ist. Der Parameter *mid* entfällt somit als Übergabe der Motor-Identifikationsnummer für die jeweiligen Funktionen.

mMoveToDistance

Syntax:	`BOOL WINAPI mMoveToDistance( double distance )`
Beschreibung:	Startet die Bewegung des Motors, der als aktueller Antrieb eingestellt ist, an die durch *distance* angegebene Motorstellung im Winkelmaß.
Parameter:	`double` *distance* - Winkelposition, an die der Motor bewegt werden soll
Rückgabewert:	`FALSE` = Fehlerfall: Die Laufzeitbibliothek `motors.dll` konnte nicht korrekt initialisiert werden, d.h. die Konfigurationsdatei wurde nicht erfolgreich eingelesen. `TRUE` = Da keine Fehlerkontrolle beim Funktionsaufruf stattfindet, ist die Funktion in jedem anderen Fall (auch bei ungültiger Winkelangabe im Parameter *distance*) erfolgreich.
Siehe auch:	`mMoveByDistance`, `mlMoveToDistance`, `mGetDistance`

mMoveByDistance

Syntax:	`BOOL WINAPI mMoveByDistance( double distance )`
Beschreibung:	Bewegt den aktuellen Antrieb relativ zur aktuellen Motorposition um eine bestimmte Distanz, die im Parameter *distance* im Winkelmaß übergeben wird.
Parameter:	`double` *distance* - Winkel, um den der Motor bewegt werden soll
Rückgabewert:	`FALSE` = Fehlerfall: Die Laufzeitbibliothek `motors.dll` konnte nicht korrekt initialisiert werden, d.h. die Konfigurationsdatei wurde nicht erfolgreich eingelesen. `TRUE` = Da keine Fehlerkontrolle bei der Funktionsausführung stattfindet, ist die Funktion in jedem anderen Fall (auch bei ungültiger Winkelangabe im Parameter *distance*) erfolgreich.
Siehe auch:	`mMoveToDistance`, `mlMoveToDistance`, `mGetDistance`

mSetLine

Syntax:	`BOOL WINAPI mSetLine( int channel , BOOL state )`
Beschreibung:	Mit dieser Funktion wird zur Zeit der Digital-Port der C-812-Steuerkarte angesprochen.
Parameter:	`int` *channel* - Nummer des Kanals, der als Ausgangsport eingestellt werden soll. Korrekte Werte für *channel* liegen zwischen 1 und 16. `BOOL` *state* - gibt an, ob der Ausgangskanal auf 0 Volt (logisch 0 $\rightarrow$ *state* = `FALSE`) oder 5 Volt (logisch 1 $\rightarrow$ *state* = `TRUE`) gesetzt werden soll.
Rückgabewert:	`TRUE` = Erfolgreiche Kommunikation über den Digital-Port der C-812-Steuerkarte

`FALSE` = Fehlerfälle:

a) Die Laufzeitbibliothek `motors.dll` konnte nicht korrekt initialisiert werden.

b) Der Parameter *channel* liegt außerhalb des zulässigen Bereichs.

c) Die notwendigen Motorkommandos konnten nicht ohne Probleme an die Motorsteuerkarte übermittelt werden.

Siehe auch: -

mIsMoveFinish

Syntax: `BOOL WINAPI mIsMoveFinish( void )`

Beschreibung: Stellt fest, ob der aktuelle Motor die anzufahrende Winkelposition erreicht hat.

Parameter: -

Rückgabewert: `TRUE` = Antrieb hat die vorgegebene Stellung angefahren und befindet sich in Ruhe.
`FALSE` = Motor ist in Bewegung oder an einer Endlage (Deathband) zum Stehen gekommen.

Siehe auch: `mlIsMoveFinish`, `mMoveToDistance`

mIsRangeHit

Syntax: `BOOL WINAPI mIsRangeHit( void )`

Beschreibung: Diese Funktion stellt fest, ob eine Bereichsbeschränkung überschritten wurde. Dazu wird der Wert der Motor-Membervariablen *bRangeHit* zurückgeliefert.

Diese Variable wird in der Umrechnungsfunktion `BOOL TMotor::Translate(long &pos,double ang)` gesetzt, falls der übergebene Winkelparameter nicht ordnungsgemäß in eine Motorpositionen innerhalb der Softwareschranken umgerechnet werden kann. Außerdem kann *bRangeHit* in der Motorfunktion `BOOL TC_832::IsLimitHit(void)` verändert werden.

Parameter: -

Rückgabewert: `TRUE` = Fehlerfall: Antrieb sollte eine Position anfahren, die außerhalb des zulässigen Bereichs liegt.
`FALSE` = Es kam zu keiner Bereichsüberschreitung.

Siehe auch: -

mIsCalibrated

Syntax: `BOOL WINAPI mIsCalibrated( void )`

Beschreibung: Stellt fest, ob für den aktuellen Motor ein gültiger Referenzpunktlauf durchgeführt wurde. Der Wert der Motor-Membervariable *bCalibrated* wird zu diesem Zweck zurückgegeben.

Parameter: -

Rückgabewert: `TRUE` = Gültiger Referenzpunktlauf für den aktuellen Antrieb ist sichergestellt.
`FALSE` = Für den aktuellen Motor existiert kein gültiger Referenzpunktlauf.

Siehe auch: -

mIsDistanceRelative

Syntax: `BOOL WINAPI mIsDistanceRelative( void )`

Beschreibung: Gibt an, ob sich die Positionsangaben (im Winkelmaß)

	des aktuellen Antriebs auf eine gesetzte Relative Null beziehen oder ob mit absoluten Angaben gearbeitet wird. Es wird dazu geprüft, ob die Motor-Membervariable *dAngleBias* Null ist.
Parameter:	-
Rückgabewert:	`TRUE` = *dAngleBias*(Winkel-Offset) $\neq$ 0, d.h. alle Winkelangaben beziehen sich auf die Relative Null. `FALSE` = Winkel-Offset *dAngleBias* = 0, d.h. alle Winkelangaben sind absolute Werte.
Siehe auch:	`mSetRelativeZero`, `mlGetOffset`, `mSetAngleDefault`

mGetDistance

Syntax:	`BOOL WINAPI mGetDistance( double & position )`
Beschreibung:	Ermittelt für den aktuellen Antrieb die aktuelle Stellung im Winkelmaß (Grad etc.). Die Position wird von der Hardware ausgelesen und wird, falls der Motor sich in Ruhe befindet und kein Übertragungsfehler aufgetreten ist, in der Motorklassenvariablen *dAngle* gespeichert und im `double` Parameter zurückgeliefert.
Parameter:	`double &` *position* - Rückgabeparameter für die ausgelesene Position
Rückgabewert:	`TRUE` = Die Position konnte ermittelt werden, wurde in der Motor-Membervariablen *dAngle* abgelegt und wird im `double&` Parameter als Winkel zurückgeliefert. `FALSE` = Fehlerfall: Motor war in Fahrt, oder die Position konnte nicht ausgelesen werden.
Siehe auch:	`mGetValue`, `mlGetValue`, `mlGetDistance`

mStopDrive

Syntax:	`void WINAPI mStopDrive( BOOL restart )`
Beschreibung:	Mit dieser Funktion wird die Bewegung des aktuellen Antriebs gestoppt. Abhängig vom Parameter *restart* wird der Motor abrupt (*restart* = `FALSE`)oder kontrolliert angehalten.
Parameter:	`BOOL` *restart* - gibt an, ob ein Neustart des Antriebs möglich sein soll (*restart* = `TRUE`) oder ob der Motor durch das Aufheben des Motorstroms sofort gestoppt werden soll (*restart* = `FALSE`).
Rückgabewert:	Obwohl die aufgerufenen Motorfunktionen ihrerseits prüfen, ob die Motorkommandos erfolgreich abgesendet werden konnten, wird der Rückgabeparameter von der `mStopDrive` Funktion nicht mehr weiterverarbeitet.
Siehe auch:	`mMoveToDistance`, `mMoveByDistance`

mGetDistanceProcess

Syntax:	`double WINAPI mGetDistanceProcess( void )`
Beschreibung:	Die Funktion ermittelt die aktuelle Motorposition des aktiven Antriebs von der jeweiligen Steuerkarte und speichert diese in der Motor-Membervariablen *lPosition* ab. Außerdem wird die Position von Encoderschritten ins Winkelmaß umgerechnet und in der Membervariablen *dAngel* abgelegt. Der Wert von *dAngel* wird als Rückgabewert geliefert.
Parameter:	-
Rückgabewert:	`double` Wert der Motor-Membervariablen *dAngle* = aktuelle Winkelposition des aktiven Antriebs
Siehe auch:	`mGetDistance`

mGetValue

Syntax: `double WINAPI mGetValue( TValueType )`

Beschreibung: Abhängig vom Wert des Parameters *vtype* wird der dazugehörige Wert der Motorgröße der aktiven Achse zurückgeliefert. Der Typ TValueType (deklariert in `comhead.h`) kann folgende Werte annehmen: *Distance* (Winkelstellung des Motors), *MinDistance*, *MaxDistance*, *Speed*, *Width* (Winkelschrittweite). Die Funktion wertet nur die Membervariablen der Motoren aus. Es findet kein Zugriff auf die Hardware statt.

Parameter: `TValueType` *vtype* - Parameter gibt an, welcher Motorwert zurückgeliefert werden soll.

Rückgabewert: `double` Wert der durch den `TValueType` Parameter bestimmten Membervariablen des aktiven Antriebs, wobei folgende Zuordnung stattfindet: *Distance* = *dAngle*, *MinDistance* = *dAngleMin*, *MaxDistance* = *dAngleMax*, *Speed* = *dSpeed* unter Berücksichtigung von *dwVelocity* bzw. *Width* = *dAngleWidth*

Siehe auch: `mGetDistance`, `mlGetValue`, `mlGetDistance`, `mSetValue`

mGetUnitType

Syntax: `TUnitType WINAPI mGetUnitType( void )`

Beschreibung: Diese Funktion ermittelt, welche Einheit für den aktuellen Motor zur Angabe von Positionen verwendet wird, indem der Wert der Motor-Membervariablen *eUnit* zurückgeliefert wird. Der Typ von eUnit ist TUnitType und in der Headerdatei `comhead.h` deklariert. Es sind die folgenden Einheiten möglich: Grad, Minuten, Sekunden, Millimeter, Mikrometer, Channel und None (keine Einheit).

Parameter: -

Rückgabewert: `TUnitType` Wert der Motor-Membervariablen *eUnit* - gibt an, in welcher Einheit die Positionsangaben des aktiven Antriebs im Steuerprogramm vorgenommen werden. Die Enumeration TUnitType ist in `comhead.h` deklariert und kann folgende Werte annehmen: Grad, Minuten, Sekunden, Millimeter, Mikrometer, Channel, None.

Siehe auch: `mGetAxisUnit`, `mGetAxisName`

mSetValue

Syntax:
```
BOOL WINAPI mSetValue( TValueType vtype,
                       double     value )
```

Beschreibung: Die Funktion setzt für den aktuellen Antrieb die Werte von Geschwindigkeit (*Speed*) oder Winkelschrittweite (*Width*). Abhängig vom Wert des Parameters *vtype* vom Typ TValueType (deklariert in `comhead.h`) wird der dazugehörige Wert der Motorgröße der aktiven Achse gesetzt. Hierzu werden nur die Membervariablen der Motoren verändert. Es findet kein Zugriff auf die Hardware statt.

Parameter: `TValueType` *vtype* - Parameter gibt an, welcher Motorwert gesetzt werden soll. Zulässig sind nur die Werte *Speed* (Geschwindigkeit) → Membervariable *dSpeed* und *Width* (Winkelschrittweite) → Membervariable *dAngleWidth*.
`double` value - zuzuweisender Wert der durch den `TValueType` Parameter bestimmten Motorgröße des aktiven Antriebs (bei *Speed* ist die Einheit des Werts Units pro Sekunde, der Wert wird automatisch durch `MaxVelocity`) beschränkt).

Rückgabewert: `TRUE` = *Speed* bzw. *Width* konnten korrekt verändert werden.

	`FALSE` = Fehlerfall: Entweder sollte ein anderer Wert als *Speed* oder *Width* gesetzt werden, oder der Wert der Winkelschrittweite lag außerhalb der Softwarebeschränkung.
Siehe auch:	`mGetValue`, `mlGetValue`

mActivateDrive

Syntax:	`void WINAPI mActivateDrive( void )`
Beschreibung:	Diese Funktion aktiviert den aktuellen Antrieb, indem die sie die Regelung der Motorsteuerkarte für diesen Motor aktiviert.
Parameter:	-
Rückgabewert:	-
Siehe auch:	`mStopDrive`

mSetCorrectionState

Syntax:	`void WINAPI mSetCorrectionState( BOOL state )`
Beschreibung:	Mit dieser Funktion wird die Korrektur bei der Umrechnung von Positionen in Winkelstellungen mit Hilfe der Funktion `TMotor::Translate` gesteuert. Ist die Korrektur aktiviert, erfolgt die Umrechnung der Motorposition von Encoderschritten in Winkel über ein Polynom 3. Grades. Andernfalls wird eine lineare Korrektur vorgenommen. Außerdem wird ermittelt, ob der Antrieb Beugung grob (DC) vorhanden ist, um die Abhängigkeiten von DC und DF (Beugung fein) auszugleichen. Da sich die Motoren DC und DF bei Bewegung gegenseitig, bedingt durch die Kopplung mittels eines gekrümmten Hebels, beeinflussen, werden die Positionen der beiden Antriebe korrigiert.

Parameter:	`BOOL` *state* - gibt an, ob die Korrektur bei der Positionsumrechnung in Winkelangaben durch ein Polynom 3. Grades (*state* = `TRUE`) oder linear (*state* = `FALSE`) erfolgen soll. Achtung: Wenn der aktuelle Antrieb nicht kalibriert ist (Membervariable *bCalibrated* = `FALSE`) oder nicht korrigiert werden kann bzw. soll (*bCorrection* = `FALSE`), setzt die Funktion den Korrekturstatus auf lineare Korrektur (*eCorrStatus* = `CorrLinear`).
Rückgabewert:	-
Siehe auch:	-

mGetAxisName

Syntax:	`LPCSTR WINAPI mGetAxisName( void )`
Beschreibung:	Liefert den Namen des aktuellen Antriebs zurück. Der Name wird bei der Initialisierung der Dynamischen Bibliothek `motors.dll` aus dem *Name*-Eintrag des Motors in der Konfigurationsdatei ausgelesen und in der TMotor-Membervariablen *szCharacteristic* gespeichert.
Parameter:	-
Rückgabewert:	`LPCSTR` Wert der Motor-Membervariablen *szCharacteristic*, in der der Name des aktuellen Antriebs aus der INI-Datei abgelegt ist.
Siehe auch:	`mGetAxisUnit`, `mlParsingAxis`, `mlGetIdByName`, `mlIsAxisValid`

mGetAxisUnit

Syntax:	`LPCSTR WINAPI mGetAxisUnit( void )`
Beschreibung:	Die Funktion liefert den Wert der TMotor-Membervariablen *szUnit* als Einheit für die Positionsangaben des aktiven Antriebs, wie sie im Steuerpro-

gramm vorgenommen werden, zurück. Die Einheit wird bei der Initialisierung der Dynamic Link Library `motors.dll` aus dem *Unit*-Eintrag des Motors in der Konfigurationsdatei ausgelesen und in *szUnit* gespeichert.

Parameter: -

Rückgabewert: `LPCSTR` Wert der Motor-Membervariablen *szUnit*, die die Einheit für die Positionsangaben des aktiven Antriebs enthält

Siehe auch: `mGetUnitType`, `mGetAxisName`

mGetDF

Syntax: `LPCSTR WINAPI mGetDF( void )`

Beschreibung: Gibt die Formatzeichenkette für die Stellengenauigkeit zur Darstellung der Motorwerte des aktuellen Antriebs im Steuerprogramm zurück.
Das Format wird mit Hilfe des *Digits*-Eintrags in der Konfigurationsdatei beim Initialisieren des Motors festgelegt und in der Motor-Membervariable *DFmt* als Zeichenkette der Form `"%.2lf"` abgespeichert. *DFmt* wird dazu verwendet, die Werte bestimmter Antriebsgrößen wie z.B. *dMinAngle*, *dSpeed*, *dDistance* etc. mit einer einheitlichen und sinnvollen Anzahl von Nachkommastellen in den Dialogen zu präsentieren.
Es wird eine Nachkommastelle weniger als bei Verwendung des Formatstrings *SFmt* berücksichtigt.

Parameter: -

Rückgabewert: `LPCSTR` Wert der Motor-Variablen *DFmt*, der bei der Initialisierung des aktuellen Antriebs abhängig vom Konfigurationsdateieintrag *Digits* mit einem Formatstring der Art `"%.2lf"` belegt wurde.

Siehe auch: `mGetSF`

mGetSF

Syntax: `LPCSTR WINAPI mGetSF( void )`

Beschreibung: Die Funktion gibt im Gegensatz zu `mGetSF` eine Formatzeichenkette für die Schrittweitenwerte des aktuellen Antriebs zur Festlegung der Darstellungsgenauigkeit im Steuerprogramm zurück.
Das Format wird mit Hilfe des *Digits*-Eintrags in der Konfigurationsdatei beim Initialisieren des Motors festgelegt und in der Motor-Membervariable *SFmt* als Zeichenkette der Form `"%.2lf"` abgespeichert. *SFmt* wird hauptsächlich dazu verwendet, die Werte der Antriebsschrittweite (Membervariable *dAngleWidth*) mit einer einheitlichen und sinnvollen Anzahl von Nachkommastellen in den Dialogen zu präsentieren.
Es wird eine Nachkommastelle mehr als bei Verwendung des Formatstrings *DFmt* berücksichtigt.

Parameter: -

Rückgabewert: `LPCSTR` Wert der Motor-Variablen *SFmt*, der bei der Initialisierung des aktuellen Antriebs abhängig vom Konfigurationsdateieintrag *Digits* mit einem Formatstring der Art `"%.2lf"` belegt wurde.

Siehe auch: `mGetDF`

mSetAngleDefault

Syntax: `void WINAPI mSetAngleDefault( void )`

Beschreibung: Setzt die Voreinstellungen für das Winkeloffset der Relativen Null und die Softwareschranken im Winkelmaß. Dazu werden die Membervariablen *dAngleBias*, *dAngleMin* und *dAngleMax* des aktuellen Antriebs auf ihre Defaultwerte zurückgestellt.

Für *dAngleMin* und *dAngleMax* werden die Membervariablen *lPositionMin* und *lPositionMax*, die die zulässigen Positionsbereiche in Encoderschritten darstellen, ausgewertet; *dAngleBias* wird auf 0.0 gesetzt.

Parameter: -

Rückgabewert: -

Siehe auch: `mlSetAngleDefault`

mSetRelativeZero

Syntax:
```
void WINAPI mSetRelativeZero( BOOL   status,
                              double offset )
```

Beschreibung: Setzt das Offset für die Relative Null des aktuellen Antriebs. Der `double` Wert des Offsets wird in der Motor-Membervariablen *dAngleBias* abgelegt. Damit wird festgelegt, ob bei Positionsangaben (im Winkelmaß) für den aktuellen Antrieb mit relativen oder absoluten Angaben gearbeitet wird.

Parameter: `BOOL` *status* - gibt an, ob die Relative Null gesetzt werden soll (→ *status* = `TRUE`) oder ob sie wieder aufgehoben werden soll. (→ *status* = `FALSE`)
`double` *offset* - der Wert des Offsets (Winkelstellung des Antriebs, an dem die Relative Null gesetzt wird), der von den absoluten Motorpositionen abgezogen wird

Rückgabewert: -

Siehe auch: `mIsDistanceRelative`, `mlGetOffset`

mExecuteCmd

Syntax: `int WINAPI mExecuteCmd( LPSTR command )`

Beschreibung: Sendet die Zeichenkette mit dem Motorkommando direkt an die zum aktuellen Antrieb gehörige Motorkarte. Es ist es aber nur vorgesehen, Motoren der Steuerkarten vom Typ C-812 anzusteuern. Wenn die Motorsteuerkarte betriebsbereit ist und es sich um ein gültiges Kommando handelt, wird dieses ausgeführt, ansonsten wird ein Fehlercode zurückgeliefert. Bei C-832-Karten wird kein Kommando ausgeführt, und die Funktion kehrt sofort mit Erfolg zurück, was sicherlich nicht die beste Lösung ist.

Parameter: `LPSTR` *command* - ein gültiges Motorkommando für die C-812-DC-Controllerkarte
Z.B.: „Move Absolute"-Kommando für C-812-Karten: `[`*boardID*`] MA [`*position*`] [RETURN]`
Dabei gibt *boardID* die vom aktuellen Antrieb belegte Achsennummer auf der Steuerkarte an (z.B. 1. Achse am C-812-Controller → *boardID*=1). Der Parameter *position* stellt die anzufahrende Position in Encoderschritten dar. Außerdem muß jedes Kommando mit dem ASCII-Code 13_{dec} (RETURN) abgeschlossen werden.

Rückgabewert: `int` Wert des Fehlercodes, der bei dem Versuch der Kommandoausführung zurückgeliefert wird. Als Fehlercodes sind folgende Werte in `comhead.h` definiert:
`R_OK` = 999 → die Übertragung und Ausführung des Kommandos konnte ohne Fehler vorgenommen werden.
`R_Failure` = 202 → der Kommandomodus der Steuerkarte ist bereits aktiv, d.h. das Kommando kann z.Z. nicht angenommen werden.
`R_TimeOut` = 214 → das Kommando konnte nicht im Zeitrahmen an die Hardware gesendet werden.
`0` → die DC-Controllerkarte ist nicht korrekt vom Steuerprogramm erkannt worden.

Siehe auch: -

mPushSettings

Syntax: `void WINAPI mPushSettings( void )`

Beschreibung: Diese Funktion dient dazu, die Werte der Motormembervariablen *dAngle*, *dAngleMin*, *dAngleMax*, *dAngleWidth* und *dSpeed* in der zum aktuellen Antrieb gehörigen Struktur *Settings* vom Typ TMSettings (deklariert in `m_motcom.h`) zwischenzuspeichern. Dabei werden eventuell früher gespeicherte Werte ohne Warnung überschrieben. Zwischen zwei `mPushSettings`-Aufrufen sollte also ein Aufruf von `mPopSettings` erfolgen, sonst gehen die zuerst gespeicherten Werte verloren.

Parameter: -

Rückgabewert: -

Siehe auch: `mPopSettings`

mPopSettings

Syntax: `void WINAPI mPopSettings( TMParameter mParam )`

Beschreibung: Schreibt die mit der Funktion `PushSettings` in der Struktur *Settings* gespeicherten Werte in die entsprechenden Member-Variablen des aktuellen Antriebs zurück. Die Struktur *Settings* ist eine Motormembervariable vom Typ TMSettings, deklariert in `m_motcom.h`. Beim Zurückschreiben werden die Werte für die Softwareschranken *dAngleMin* und *dAngleMax*, die Winkelschrittweite *dAngleWidth* und die Motorgeschwindigkeit *dSpeed* geändert. Abhängig vom übergebenen Parameter *mParam* vom Typ `TMParameter` (deklariert in `comhead.h`) besteht die Möglichkeit, die alte Winkelstellung, d.h. den alten Wert von *dAngle*, wiederherzustellen.

Parameter: `TMParameter` *mParam* - diese Enumeration ist in

`comhead.h` deklariert und kann folgende Werte annehmen: *ThisPosition*, *OldPosition*, *WaitExecution*, *NoWait*. Die Funktion `mPopSettings` testet aber nur, falls sich der Motor nicht mehr an der alten gespeicherten Winkelstellung befindet, ob *mParam* den Wert *OldPosition* hat. Dann wird die alte Motorposition angefahren.

Rückgabewert: -

Siehe auch: `mPushSettings`

mStartMoveScan

Syntax: `void WINAPI mStartMoveScan( int nTimeTicks, int )`

Beschreibung: Startet einen Scan zum Optimieren des Anfahrverhaltens der Antriebe C-812ISA, C-812GPIB und C-832. Aufgerufen wird diese Funktion vom Dialog *Einstellungen/Antriebe/Optimieren...* aus dem Hauptfenster des RTK-Steuerprogramms. Die Funktion initialisiert zunächst den Speicher für die Scan-Daten. Die Anzahl der Meßwerte ist durch das Steuerprogrammm vorgegeben. Anschließend wird je nach aktuellem Antrieb der Scan gestartet. Dies geschieht mit der Funktion `StartCheckScan` die den Antrieb um *wPositionWidth* relativ zur aktuellen Position fährt. Das heißt, es wird die Position „aktuelle Position"+ *wPositionWidth* angefahren. Die Positionen werden in Encoderschritten angegeben. Danach wird ein Timer aktiviert, der durch sein Auslösen die Funktion `mSavePosition` aufruft. Je nach Antrieb wird nach entsprechenden Zeitintervallen die Funktion `mSavePosition` gestartet.

Parameter: Der 1. Parameter gibt an, in welchen Zeitabständen der Timer aufgerufen wird, der in `StartCheckScan` gestartet wurde. Die Einheit wird in Millisekunden angegeben.

Der 2. Parameter ist unwichtig. Wahrscheinlich war die Entwicklung noch nicht abgeschlossen und deshalb hat der 2. Parameter keine Funktion. Wichtig ist nur, daß dieser Parameter beim Aufruf von `mStartMoveScan` nicht vergessen wird. Sinnvoll ist es ihn mit 0 zu initialisieren.

Rückgabewert: -

Siehe auch: `mlOptimizingDlg`, `mGetMoveScan`, `mGetScanSize`, `mSavePosition`

mSavePosition

Syntax: `void CALLBACK mSavePosition(UINT,UINT,DWORD,DWORD, DWORD)`

Beschreibung: Funktion die aufgerufen wird, wenn ein Timer während eines durch `mStartMoveScan` gestarteten Scans ausgelöst wurde. Die Funktion `StartCheckScan` aktiviert den Timer und gibt als Bearbeitungfunktion `mSavePosition` an. Je nach ausgewähltem Motor wird in einem bestimmten Zeitintervall diese Funktion durch den Timer aufgerufen. Sie speichert dann sukzessive die aktuelle Position in einem Datenfeld ab. Wenn der Motor stoppt, bevor das Scanfile vollständig mit Meßwerten gefüllt ist, wird der *MoveFinishIdx* gesetzt. Dieser kennzeichnet die letzte Position, an der der Motor in Bewegung war. Wurden alle Meßwerte erfaßt und der Motor befindet sich noch nicht im Stillstand, so wird der *MoveFinishIdx* auf 0 gesetzt. Am Ende des Scans wird der Timer wieder deaktiviert und eine Nachricht, daß der Scan abgeschlossen wurde, in die Nachrichtenschleife verschickt.

Parameter: Der 1. Parameter ist die ID des Timers der diese Callback-Funktion aufruft. Über diese ID kann der Timer identifiziert und wenn er nicht mehr benötigt wird, gelöscht werden. Die restlichen Parameter haben in diesem Fall keine Funktion.

Rückgabewert:	-
Siehe auch:	`mStartMoveScan`, `mGetMoveScan`, `mGetMoveFinishIdx`

mGetMoveScan

Syntax:	`LPLONG WINAPI mGetMoveScan( void )`
Beschreibung:	Diese Funktion übergibt die Daten, die beim Scan von `mStartMoveScan` zum Optimieren der Dialoge erfaßt wurden. Dabei handelt es sich um ein Feld von Positionsdaten, die in einem Fenster visualisiert werden.
Parameter:	-
Rückgabewert:	Pointer auf ein Datenfeld mit LONG-Elementen, die die ermittelten Positionsdaten des Checkscans enthalten.
Siehe auch:	`mStartMoveScan`, `mGetScanSize`

mGetScanSize

Syntax:	`int WINAPI mGetScanSize( void )`
Beschreibung:	Gibt die Größe des Datenfeldes an, in dem die Meßwerte des Scans stehen, der in `mStartMoveScan` gestartet wurde.
Parameter:	-
Rückgabewert:	Die Größe des Scans ist vom Steuerprogramm festgelegt auf *nScanSize*=150 Werte.
Siehe auch:	`mStartMoveScan`, `mGetMoveScan`

mGetMoveFinishIdx

Syntax:	`int WINAPI mGetMoveFinishIdx( void )`
Beschreibung:	Der Wert *MoveFinishIdx* wird in `mSavePosition` ermittelt. Er gibt an, bei welchem Meßwert im Datenfeld des Scans der betrachtete Motor seine Bewegung beendet hatte. War der Motor nach Beenden des Scans noch in Bewegung, so erhält *MoveFinishIdx* den Wert 0.
Parameter:	-
Rückgabewert:	Der Rückgabewert enthält den Index an dem der Motor den Stillstand erreicht hatte.
Siehe auch:	`mStartMoveScan`, `mSavePosition`

Index