DESIGNBESCHREIBUNG NEUE ‚MANUELLE JUSTAGE'

Dokument zur Studienarbeit
- Designphase -

Autoren	Thomas Kullmann, Günther Reinecker
Dokumentversion	2.0
Zustand	abgeschlossen
letzte Bearbeitung	25.01.04

Inhaltsverzeichnis

I Überblick

Die ursprüngliche ‚Manuelle Justage' zeigt nach eingehender Bewertung einige Mängel. Diese reichen von einer schlechten Ergonomie der Oberfläche (siehe [2]) bis zu konkreten Designfehlern (siehe [3]). Da die ‚Manuelle Justage' ein wichtiger Teil des XCtl-Projekts ist, erfolgt nun ein Neuentwurf.

Das Dokument beinhaltet eine vollständige Auflistung der verwendeten Klassen, Strukturen, Attribute und Methoden, die bei der Implementation der Funktionalität oder dem Oberflächenentwurf Verwendung finden. Diese wurde nach selbst definierten Layoutkonventionen formalisiert (siehe [10])**.** Die einzelnen Elemente sind fett hervorgehoben und werden jeweils kurz erläutert, wobei auch auf Zusammenhänge untereinander hingewiesen wird. Bei grundlegenden Verständnisproblemen sollte das zugehörige Pflichtenheft ([1]) mit hinzugezogen werden. Direkte Umsetzungen von Pflichtenheftfunktionen sind explizit vermerkt.

Die wichtigste Eigenschaft dieses Neuentwurfs ist die klare Trennung von Funktionalität (**Abbildung 1**, türkis gefärbte Klassen) und Oberfläche (**Abbildung 1**, gelb gefärbte Klassen). Das Design soll einen problemlosen Austausch der Oberfläche ermöglichen, ohne die Implementation der Funktionalität zu verändern.

Zur Einhaltung der Grundprinzipien für die Anbindung einer grafischen Nutzeroberfläche an eine Funktionskomponente (siehe [4]) wird im Konstruktor der Dialogfensterklasse ein Objekt der Funktionskomponente erzeugt, über das die gesamte Verwaltung und Steuerung der Antriebe und die Halbwertsbreitenmessung abrufbar ist. Zur Überwachung dieser Vorgänge muss die Oberfläche in regelmäßigen Abständen den Fortschritt bei der Funktionskomponente anfragen. Dafür ist die Timer-Komponente (grün) erforderlich, die in 100ms-Abständen Ereignisse erzeugt. Dazu wird ein `TInterfaceTimer`-Objekt im Konstruktor der Hauptdialog-Fensterklasse erzeugt und eine Referenz auf das Fenster übergeben, damit dieses Fenster über die Timerereignisse informiert wird.

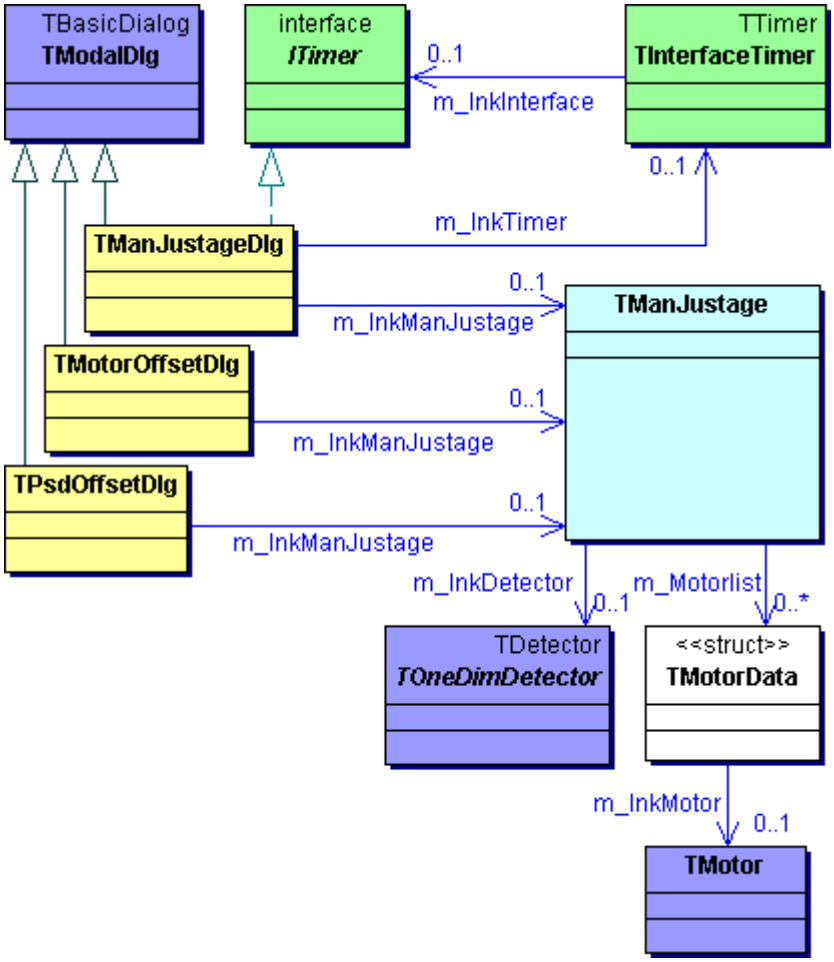

Abbildung 1 Klassendiagramm des Neuentwurfs der Manuellen Justage

I.1 Hinweise zur Benennung

Hier sind die Konventionen zur Benennung und Formatierung von Attributen und Methoden aufgelistet, wie sie durchweg im gesamten Entwurf konsequent angewendet wurden.

Klassennamen:

- Funktionalität: `T`<Name>
- Oberfläche: `T`<Name>`Dlg`

Benennung von Methoden:

- M., die eine Aktion auslösen, beginnen mit `Do`.
- M., die den Inhalt von Attributen zurückgeben, beginnen mit `Get`.
- M., die den Inhalt von Attributen schreiben, beginnen mit `Set`.
- M., die Ereignisse darstellen beginnen mit `On`.
- M., die zurückgeben, ob eine `Do`- bzw. `Set`-Methode aufgerufen werden kann, beginnen mit `Can`.

Benennung der Attribute:

- A. beginnen stets mit `m_`, wobei der folgende Buchstabe groß geschrieben wird.

II Strukturen

typedef enum { mtDirect, mtDrive, mtStep } EMotion- **GLOBAL**

ist der Aufzählungstyp für die Betriebsarten.

Betriebsart	Beschreibung
`mtDirect`	Direkt-
`mtDrive`	Fahr-
`mtStep`	Schrittbetrieb

Tabelle 1 Beschreibung der Elemente von `EMotionType`

typedef enum { dUp, dDown } Edirection **GLOBAL**

ist der Aufzählungstyp für die Bewegungsrichtungen.

Bewegungsrichtung	Beschreibung
`dUp`	Vorwärts-
`dDown`	Rückwärtsbetrieb

Tabelle 2 Beschreibung der Elemente von `EDirection`

typedef double TAngle **GLOBAL**

ist der Typ für Antriebspositionen (**in Nutzereinheiten**).

typedef double Tspeed **GLOBAL**

ist der Typ für Antriebsgeschwindigkeiten (**in Nutzereinheiten/Sekunde**).

typedef int Tchannel **GLOBAL**

ist der Typ für Detektor-Kanalwerte.

const TAxisType _HWBAXIS= DF **GLOBAL**

Ist der Typ des Antriebs, der für die Halbwertsbreitenmessung verwendet wird (siehe STANDARD.MAK, INQUIREHWB-Abschnitt).

III Funktionalität

Abbildung 2 Klassendiagramm der Funktionalität zur neuen ‚Manuellen Justage'

III.1 Klasse TManJustage

Deklaration : MJ_FUNK.H
Implementation: MJ_FUNK.CPP

III.1.1 Strukturen

- **`TMotorData[] m_MotorList`** **PRIVATE**

ist die Liste der in der ‚Manuelle Justage' verfügbaren Antriebe mit Zusatzinfos und wird im Konstruktor initialisiert.

- **`typedef struct TMotorData`** **PRIVATE**

In dieser Struktur sind alle Informationen eines Antriebs zusammengefasst. Jede Instanz dieser Klasse kapselt die Daten für jeweils genau einen Antrieb der ‚Manuellen Justage'.

Attribute in `TMotorData`		Verwendung
`TMotor *m_lnkMotor`	PUBLIC	ist die Referenz auf den entsprechenden Antrieb (siehe [1], Daten: für diverse / **D <dd>** / des Antriebs).
`TAxisType m_AxisType`	PUBLIC	ist der Antriebstyp.
`char m_MotorId[MaxString]`	PUBLIC	beinhaltet die Bezeichnung der Sektion innerhalb der HARDWARE.INI - Muster „MOTOR<M>".
`EMotionType m_MotionType`	PUBLIC	ist die aktuelle Betriebsart des Antriebs. Dieser Wert wird bei der Initialisierung aus der HARDWARE.INI –> [MOTOR<M>] –> MJ_MotionType gelesen, ansonsten im Dialogfenster gesetzt (siehe [1], Daten: / **D 100** /).
`TAngle m_AngleDest`	PUBLIC	ist die Zielposition (**in Nutzereinheiten**) für den Direktbetrieb. Dieser Wert wird bei der Initialisierung aus der HARDWARE.INI –> [MOTOR<M>] –> MJ_AngleDest gelesen, ansonsten im Dialogfenster gesetzt (siehe [1], Daten: / **D 110** /).
`TSpeed m_Speed`	PUBLIC	ist der Wert für die Geschwindigkeit im Direkt- und Fahrbetrieb (**in Nutzereinheiten/ Sekunde**). Dieser Wert wird bei der Initialisierung aus HARDWARE.INI –> [MOTOR<M>] –> MJ_Speed gelesen, ansonsten im Dialogfenster gesetzt (siehe [1], Daten: / **D 120** /).
`TAngle m_AngleWidth`	PUBLIC	ist der Wert für die Schrittweite im Schrittbetrieb (**in Nutzereinheiten**). Dieser Wert wird bei der Initialisierung aus der HARDWARE.INI –> [MOTOR<M>] –> MJ_AngleWidth gelesen, ansonsten im Dialogfenster gesetzt (siehe [1], Daten: / **D 150** /).
`BOOL m_KzMoving`	PUBLIC	Dieser Wert speichert, welchen Status die Nutzeroberfläche über die Bewegung des Antriebs hat. Wenn dieses Attribut angibt, dass sich der Antrieb bewegt, das Auslesen beim Antrieb jedoch Stillstand ergibt, heißt das, dass der Antrieb seit der letzten Abfrage gestoppt hat.

Tabelle 3 Beschreibung der Elemente von `TMotorData`

III.1.2 Attribute

TMotorData *m_MotorList **PRIVATE**

Dies ist die Liste der verfügbaren Antriebe - wird bei der Initialisierung gefüllt.

TOneDimDetector *m_lnkDetector **PRIVATE**

ist der Zeiger auf einen 1-dimensionalen Detektor für das Kanaloffset - wird im Konstruktor initialisiert (siehe [1], Daten: / **D 190** / bis / **D 210** /).

int m_Channel **PRIVATE**

ist der gemessene Kanal des PSD-Detektors und wird in der DEVELOP.INI –> [Steuerprogramm] –> PSDChannel gespeichert (siehe [1], Daten: / **D 180** /).

UINT m_MotorCount **PRIVATE**

ist die Anzahl der Motoren in `m_MotorList.`

BOOL m_KzMeasuring **PRIVATE**

Wenn bei `OnTimer()` `m_IsMeasuring != IsMeasuring()` ist, dann hat die Makroverarbeitung gestartet oder gestoppt.

BOOL m_IsMeasuring **PRIVATE**

zeigt den Status der Halbwertbreitenmessung an.

III.1.3 Methoden

TManJustage(void) **PUBLIC**

ist der Standard-Konstruktor. Er initialisiert `m_MotorList` mit den aktuell angeschlossenen Antrieben und diese mit den zuletzt verwendeten Betriebsparametern und Offsets. Er testet ob ein 1-dimensionaler Detektor angeschlossen ist und initialisiert dann `m_lnkDetektor.`

~TManJustage(void) **PUBLIC**

ist der Standard-Destruktor und gibt `m_MotorList` wieder frei.

BOOL DoDirect(int) **PUBLIC**

löst eine Antriebsbewegung im Direktbetrieb aus. Der erste Parameter ist der Index des Antriebs in der Antriebsliste - setzt `m_IsMoving` (siehe [1], Funktion: / **F 70** /).

BOOL DoDrive(int, EDirection) **PUBLIC**

löst eine Antriebsbewegung im Fahrbetrieb aus. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist die Bewegungsrichtung - setzt `m_IsMoving` (siehe [1], Funktionen: / **F 90** /, / **F 100** /).

BOOL DoStep(int, EDirection) **PUBLIC**

löst eine Antriebsbewegung im Schrittbetrieb aus. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist die Bewegungsrichtung - setzt `m_IsMoving` (siehe [1], Funktionen: / **F 110** /, / **F 120** /).

BOOL DoStop(int) PUBLIC
stoppt die Antriebsbewegung unabhängig von der Betriebsart. Der Parameter ist der Index des Antriebs in der Antriebsliste (siehe [1], Funktion: / **F 80** /).

BOOL DoStopEverything(void) PUBLIC
dient dem Stoppen aller Antriebe, auch bei laufender Halbwertsbreiten- und Detektormessung.

BOOL IsIndexValid(int) PUBLIC
überprüft, ob der im Parameter übergebene Index einem Eintrag in der Antriebsliste entspricht.

BOOL HasOffset(int, BOOL&) PUBLIC
zeigt im zweiten Parameter an, ob ein Offset gesetzt ist, d.h. `Offset != 0`. Der erste Parameter ist der Index des Antriebs in der Antriebsliste. Bei gültigem Index ist der Rückgabewert == `TRUE`, sonst `FALSE`.

BOOL IsMoving(int, BOOL&) PUBLIC
zeigt im zweiten Parameter an, ob sich ein Antrieb bewegt. Der erste Parameter ist der Index des Antriebs in der Antriebsliste - gibt `m_isMoving` zurück.

BOOL IsCalibrated(int, BOOL&) PUBLIC
zeigt im zweiten Parameter an, ob ein Antrieb kalibriert ist. Der erste Parameter ist der Index des Antriebs in der Antriebsliste.

EMotionType GetMotionType(int, BOOL&) PUBLIC
gibt die aktuelle Betriebsart (`m_MotionType`) eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

TAngle GetAngleDest(int, BOOL&) PUBLIC
… gibt die aktuelle Sollposition (`m_AngleDest`) für den Direktbetrieb eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

TSpeed GetSpeed(int, BOOL&) PUBLIC
gibt die aktuelle Geschwindigkeit (`m_Speed`) für den Direkt- oder Fahrbetrieb eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

TAngle GetAngleWidth(int, BOOL&) PUBLIC
gibt die aktuelle Schrittweite (`m_AngleWidth`) für den Schrittbetrieb eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

BOOL GetKzMoving(int, BOOL&) PUBLIC
gibt `m_KzMoving` zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

TAngle GetOffset(int, BOOL&) PUBLIC
gibt das aktuelle Offset eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

UINT GetMotorCount(void) PUBLIC
gibt die aktuelle Anzahl der Antriebe (`m_MotorCount`) in der Antriebsliste zurück (siehe [1], Funktion: / **F 10** /).

LPCSTR GetMotorName(int, BOOL&) PUBLIC
gibt den Namen eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

int GetMotorIdx(LPCSTR) PUBLIC
gibt den Index eines gegebenen Antriebs zurück. Der erste Parameter ist der Name des Antriebs. Ist dieser ungültig wird `-1` zurückgegeben.

int GetMotorIdx(TAxisType) PUBLIC
gibt den Index eines gegebenen Antriebs zurück. Der erste Parameter ist die standardisierte Antriebsbezeichnung des Antriebs. Ist diese ungültig wird `-1` zurückgegeben.

TAngle GetAngle(int, BOOL&) PUBLIC
gibt die minimale Antriebsposition zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

TAngle GetAngleMin(int, BOOL&) PUBLIC
gibt die maximale Antriebsposition zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

TAngle GetAngleMax(int, BOOL&) PUBLIC
gibt die aktuelle Antriebsposition zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

UINT GetDigits(int, BOOL&) PUBLIC
gibt die Anzahl der Nachkommastellen für alle Daten eines Antriebs zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

static UINT ParsingAxis(LPCSTR) PUBLIC
liest die standardisierte Antriebsbezeichnung als Parameter und gibt den Index des Antriebs in der Antriebsliste zurück.

LPCSTR GetUnit(int, BOOL&) PUBLIC
gibt die Einheit der Antriebsdaten zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

BOOL SetMotionType(int, EMotionType&) PUBLIC
setzt die Betriebsart (`m_MotionType`) für einen Antrieb. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist die Betriebsart. Wenn das Setzen erfolgreich war, dann Rückgabewert `== TRUE`.

BOOL SetAngleDest(int, TAngle&) PUBLIC
passt die Sollposition ins Intervall [AngleMin, AngleMax] und setzt danach m_DestPos für den Direktbetrieb eines Antriebs. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist die Sollposition. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE.

BOOL SetSpeed(int, TSpeed&) PUBLIC
passt die Geschwindigkeit ins Intervall [MinimalSpeed, MaximalSpeed] und setzt danach m_Speed für den Fahrbetrieb und Direktbetrieb eines Antriebs. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist die Geschwindigkeit. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE.

BOOL SetAngleWidth(int, TAngle&) PUBLIC
passt die Schrittweite ins Intervall [MinimalWith, MaximalWith] und setzt danach m_StepWidth für den Schrittbetrieb eines Antriebs. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist die Schrittweite. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE.

BOOL SetKzMoving(int, BOOL) PUBLIC
setzt m_KzMoving für einen Antrieb. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist der neue Wert. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE.

BOOL SetRelativeZero(int) PUBLIC
setzt die relative Null. Der Parameter ist der Index des Antriebs in der Antriebsliste. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE (siehe [1], Funktion: / **F 30** /).

BOOL ResetRelativeZero(int) PUBLIC
hebt die relative Null auf, d.h. das Offset = 0. Die angezeigte Istposition entspricht danach wieder der absoluten Antriebsposition. Der Parameter ist der Index des Antriebs in der Antriebsliste. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE (siehe [1], Funktion: / **F 40** /).

BOOL SetOffset(int, TAngle&) PUBLIC
setzt das Offset für <Antrieb>. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite ist das neue Offset. Wenn das Setzen erfolgreich war, dann Rückgabewert == TRUE.

TAngle CalcOffsetFromAngle(int, TAngle&) PUBLIC
Die Funktion gibt das Offset für <Antrieb> zwischen der absoluten Position eines Antriebs und einer neu eingegeben Position zurück. Der erste Parameter ist der Index des Antriebs, der Zweite ist die einzugebende Position (siehe [1], Funktion: / **F 50** /).

TAngle CalcAngleFromOffset(int, TAngle&) PUBLIC
Die Funktion gibt die neue Istposition eines Antriebs bei einem neu gesetzten Offset für <Antrieb> zurück. Der erste Parameter ist der Index des Antriebs, der Zweite ist das Offset für <Antrieb> (siehe [1], Funktion: / **F 60** /).

int GetDetectorAxisIdx(void) PUBLIC
gibt den Index des Antriebs für den 1-dimensionlen Detektor zurück. Ist keiner vorhanden, wird -1 zurückgegeben.

LPCSTR GetDetectorName(BOOL&) PUBLIC
gibt die Bezeichnung des 1-dimensionlen Detektors zurück, z.B. "PSD". Ist keiner vorhanden, wird der Parameter == `FALSE`, sonst `TRUE`.

BOOL HasDetectorAxis(void) PUBLIC
überprüft ob ein 1-dimensionalen Detektor angeschlossen ist.

int GetChannel(void) PUBLIC
gibt den aktuellen Kanal (`m_Channel`) des aktuellen 1-dimensionlen Detektors zurück.

TAngle GetAnglePerChannel(void) PUBLIC
gibt das aktuellen Verhältnis: Winkel pro Kanal zurück.

BOOL SetChannel(int&) PUBLIC
setzt den aktuellen Kanal (`m_Channel`) des aktuellen 1-dimensionlen Detektors. Der Parameter ist der neue Kanal. Wenn das Setzen erfolgreich war, dann Rückgabewert == `TRUE` (siehe [1], Funktion: / **F 150** /).

TAngle CalcChannelOffset(TChannel&) PUBLIC
berechnet zu einem Detektor-Kanal das entstehende Offset zur aktuellen Position des Antriebs für den 1-dimensionlen Detektor. Der Parameter ist der neue Kanal.

TAngle CalcChannelAngle(TChannel&) PUBLIC
berechnet zu einem Detektor-Kanal die neu anzuzeigende Position des Antriebs für den 1-dimensionlen Detektor. Der Parameter ist der neue Kanal.

BOOL SetChannelOffset(TAngle&) PUBLIC
setzt das Kanaloffset im aktuellen 1-dimensionlen Detektor. Der Parameter ist das neue Offset. Wenn das Setzen erfolgreich war, dann Rückgabewert == `TRUE`.

BOOL ResetSetChannelOffset(int) PUBLIC
setzt das Kanaloffset = 0 im aktuellen 1-dimensionlen Detektor. Der Parameter ist das neue Offset. Wenn das Setzen erfolgreich war, dann Rückgabewert == `TRUE`.

BOOL DoStartMeasure(HWND) PUBLIC
startet eine Messung der Halbwertsbreite und setzt `m_IsMeasuring` == `TRUE`. Der Parameter ist ein Handle auf das Dialogfenster der neuen ‚Manuellen Justage' (siehe [1], Funktion: / **F 130** /).

BOOL DoStopMeasure(void) PUBLIC

stoppt die laufende Messung der Halbwertsbreite und setzt `m_IsMeasuring == FALSE`. Der abgebrochene Messvorgang kann nicht fortgesetzt werden und es wird kein Ergebnis zurückgegeben (siehe [1], Funktion: / **F 140** /).

BOOL IsMeasuring(void) PUBLIC

gibt `m_IsMeasuring` zurück und zeigt so an, ob eine Halbwertsbreitenmessung durchgeführt wird.

BOOL IsMeasureReset(void) PUBLIC

zeigt an, ob eine Halbwertsbreitenmessung abgebrochen wurde.

BOOL GetKzMeasuring(void) PUBLIC

gibt `m_KzMeasuring` zurück. Der erste Parameter ist der Index des Antriebs in der Antriebsliste, der Zweite zeigt, ob der Rückgabewert gültig ist.

BOOL GetMeasureProgress(int&, LPSTR, int&) PUBLIC

gibt Index und Anzahl der Kommandos sowie Statusinformationen bzw. den Kommandoname per Parameter zurück. Wenn eine Messung aktiv ist, dann Rückgabewert `== TRUE`.

double GetMeasureHwb(void) PUBLIC

gibt den Wert der Halbwertsbreite, wenn diese vorher gemessen wurde, oder 0.0 zurück.

TDetector *GetDetector (void) PUBLIC

gibt den Zeiger für den ausgewählten Detektor, welcher für die Halbwertsbreitenmessung benötigt wird, zurück.

void SetKzMeasuring(BOOL) PUBLIC

setzt `m_KzMeasuring`, weil `OnTimer()` auf Start/ Stop der Makroverarbeitung reagiert hat.

void UpdateDetector (void) PUBLIC

informiert die Makroverarbeitung, dass neue Messwerte vorliegen.

void CanDoDirect(int) PUBLIC

zeigt an, ob eine Antriebsbewegung im Direktbetrieb möglich ist. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanDoDrive(int) PUBLIC

zeigt an, ob eine Antriebsbewegung im Fahrbetrieb möglich ist. Der Parameter ist der Index des Antriebs.

void CanDoStep(int) PUBLIC

zeigt an, ob eine Antriebsbewegung im Schrittbetrieb möglich ist. Der Parameter ist der Index des Antriebs.

void CanDoStop(int) PUBLIC

zeigt an, ob eine Antriebsbewegung gestoppt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanSetMotionType(int) PUBLIC
zeigt an, ob die Betriebsart gesetzt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanSetAngleDest(int) PUBLIC
zeigt an, ob die Sollposition gesetzt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanSetSpeed(int) PUBLIC
zeigt an, ob die Geschwindigkeit gesetzt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanSetAngleWidth(int) PUBLIC
zeigt an, ob die Schrittweite gesetzt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanResetRelativeZero(int) PUBLIC
zeigt an, ob das Offset = 0 gesetzt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanSetOffset(int) PUBLIC
zeigt an, ob das Offset für <Antrieb> gesetzt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanDoStartMeasure(int) PUBLIC
zeigt an, ob die Halbwertsbreitenmessung gestartet werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

void CanDoStopMeasure(int) PUBLIC
zeigt an, ob die Halbwertsbreitenmessung gestoppt werden kann. Der Parameter ist der Index des Antriebs in der Antriebsliste.

III.1.4 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚Lines **O**f **C**ode' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	986
	‚**LOC** of **I**mplementation'*	LOCI	842
	‚**LOC** of **D**eclaration'*	LOCD	144
	‚**N**umber **O**f **A**ttributes'	NOA (0, 30)	8
	‚**N**umber **O**f **O**perations'	NOO (0, 50)	**62**
	‚**N**umber **O**f **M**embers' Attribute + Methoden	NOM = NOA + NOO	70
	‚**N**umber **O**f **C**onstructors'	NOCON (0, 5)	1
	‚**N**umber **O**f **O**verridden Methods'	NOOM (0, 10)	0
	‚**P**ercentage of **Priv**ate **M**embers'	PPrivM	10
	‚**P**ercentage of **Prot**ected **M**embers'	PProtM (0, 10)	0
	‚**P**ercentage of **Pub**lic **M**embers'	PPubM	90
	‚**W**eighted **M**ethods **P**er **C**lass'	WMPC1 (0, 30)	
Attribute			
	‚**A**ttribute **C**omplexity'	AC	56
Methoden			
	‚**M**aximum **N**umber **O**f **P**arameters'	MNOP (0, 4)	3
	‚**C**yclomatic **C**omplexity'	CC	
Kommentare			
	‚**N**umber **O**f **C**omments'*	NOC	382
	‚**T**rue **C**omment **R**atio'	TCR (5, 400)	57

Tabelle 4 Ausgewählte Metriken der Klasse `TManJustage` (Quelle: Together ‚Version 5.5)

* Diese Metriken sind nicht Bestandteil von Together, sondern wurden manuell ermittelt.

IV Oberfläche

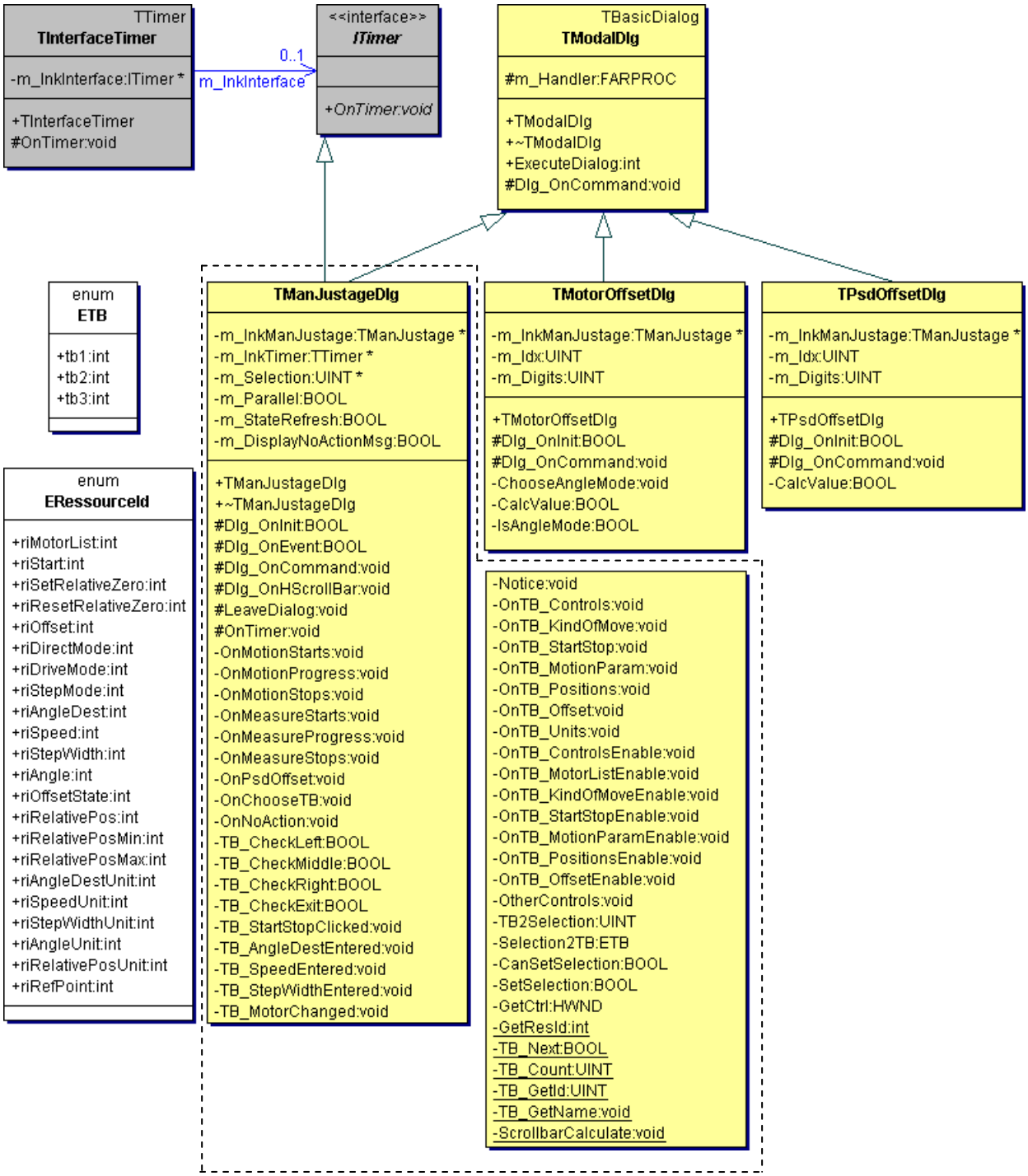

Abbildung 3 Klassendiagramm der Oberfläche zur Manuellen Justage

IV.1 Klasse TManJustageDlg

Deklaration : MJ_GUI.H

Implementation: MJ_GUI.CPP

Die Klasse `TManJustageDlg` ist von der Basisklasse für modale Dialogfenster `TModalDlg` abgeleitet.

IV.1.1 Strukturen

`typedef enum { tb1, tb2, tb3 } ETB` **GLOBAL NEU**

ist der Aufzählungstyp zur Adressierung der Teilbereiche (zu deren Verwaltung siehe Methode `TB_Next`, `TB_Count`, `TB_GetId`, `TB_GetName`).

Bewegungsrichtung	Beschreibung
`tb1`	1. Teilbereich
`tb2`	2. Teilbereich
`tb3`	3. Teilbereich

Tabelle 5 Beschreibung der Elemente von `ETB`

`typedef enum { … } ERessourceId` **GLOBAL NEU**

ist der Aufzählungstyp zur Adressierung aller Steuerelemente in den Teilbereichen. In Zusammenhang mit `ETB` kann mit Methode `GetResId` die **gleichnamige** Ressourcen-ID und mit `GetCtrl` das Fensterhandle aller teilbereichs-internen Steuerelemente ermittelt werden.

Zu adressierendes Steuerelement des Teilbereichs	Ressourcen-ID (siehe Tabelle 10)
`riMotorList`	id_p1_MotorList
`riStart`	cm_p1_Start
`riSetRelativeZero`	cm_p1_SetRelativeZero
`riResetRelativeZero`	cm_p1_ResetRelativeZero
`riOffset`	cm_p1_Offset
`riDirectMode`	id_p1_DirectMode
`riDriveMode`	id_p1_DriveMode
`riStepMode`	id_p1_StepMode
`riAngleDest`	id_p1_AngleDest
`riSpeed`	id_p1_Speed
`riStepWidth`	id_p1_StepWidth
`riAngle`	id_p1_Angle
`riOffsetState`	id_p1_OffsetState
`riRelativePos`	id_p1_RelativePos
`riRelativePosMin`	txt_p1_RelativePosMin
`riRelativePosMax`	txt_p1_RelativePosMax
`riAngleDestUnit`	txt_p1_AngleDestUnit
`riSpeedUnit`	txt_p1_SpeedUnit
`riStepWidthUnit`	txt_p1_StepWidthUnit
`riAngleUnit`	txt_p1_AngleUnit
`riRelativePosUnit`	txt_p1_RelativPosUnit
`riRefPoint`	txt_p1_RefPoint

Tabelle 6 Beschreibung der Elemente von `ERessourceId` am Beispiel des ersten Teilbereichs

IV.1.2 Attribute

TManJustage *m_lnkManJustage **PRIVATE NEU**
ist die Referenz auf die Funktionalität, wird dynamisch im Konstruktor erzeugt und im Destruktor freigegeben. Das Objekt enthält die gesamte Funktionalität zur Steuerung der ‚Manuellen Justage'.

TTimer *m_lnkTimer **PRIVATE NEU**
ist die Referenz auf den Timer, der im Konstruktor dynamisch erzeugt und im Destruktor freigegeben wird. Sie dient zur Überwachung der Antriebsbewegungen und der Halbwertsbreitenmessung. Der Timer wird nur gestartet, wenn eine zu überwachende Aktion gestartet wird und bleibt aktiv bis alle Vorgänge gestoppt werden (Auswertung durch Methode `OnTimer`).

UINT *m_Selection **PRIVATE NEU**
ist eine dynamische Liste deren Größe von der Anzahl der Teilbereiche (Methode `TB_Count`) abhängt. Sie wird im Konstruktor erzeugt, im Destruktor freigegeben und enthält für jeden Teilbereich welcher Antrieb (Index in `lpMLIST`) ausgewählt ist.

Initialisierung in `Dlg_OnCommand`: DEVELOP.INI –> [ManualAdjustment] –> <TBNAME>, wobei <TBNAME> dem Rückgabewert des zweiten Parameters von `TB_GetName` entspricht, z.B. „Section1". Der gelesene Wert ist die Bezeichnung des Antriebs, die in den Index übersetzt wird (die Reihenfolge könnte sich geändert haben). Wenn der Antrieb nicht mehr vorhanden ist, wird der erste verfügbare Antrieb (ggf. ‚kein Antrieb') ausgewählt. Wenn zuvor ‚kein Antrieb' ausgewählt war, steht anstatt der Antriebsbezeichnung „–".

BOOL m_Parallel **PRIVATE NEU**
bestimmt, ob die Nutzeroberfläche während der Aktualisierung des Inhaltes und Zustandes von Steuerelementen blockiert ist (`FALSE`) oder ob die Aktualisierung im Hintergrund stattfindet (`TRUE`) – siehe Methode `Notice` (DEVELOP.INI –> [ManualAdjustment] –> Parallel).

BOOL m_StateRefresh **PRIVATE NEU**
bestimmt, ob die Zustände der Steuerelemente (freigegeben bzw. gesperrt) bei jeder Zustandsänderung aktualisiert werden - `TRUE` = Aktualisierung, `FALSE` = Steuerelemente stets freigeben (DEVELOP.INI –> [ManualAdjustment] –> StateRefresh).

BOOL m_DisplayNoActionMsg **PRIVATE NEU**
Bei stets freigegebenen Steuerelementen (`m_StateRefresh==FALSE`) werden ungültige Nutzeraktionen von der Funktionskomponente ignoriert. Um dem Anwender ein Meldungsfenster (siehe Methode `OnNoAction`) anzuzeigen, ist dieses Attribut `TRUE` zu setzen (DEVELOP.INI –> [ManualAdjustment] –> DisplayNoActionMsg).

IV.1.3 Methoden

TManJustageDlg(void) **PUBLIC NEU**
Dieser Konstruktor dient zur Vorbereitung der Anzeige des dazugehörigen Dialogfensters. Dazu wird der Konstruktor der Basisklasse `TModalDlg` (siehe [7]) gerufen. Zudem werden die dynamischen Attribute erzeugt und alle Attribute werden initialisiert (siehe IV.1.2), wobei ein Teil mit Werten aus der DEVELOP.INI gefüllt wird.

virtual ~TManJustageDlg(void) **VIRTUAL PUBLIC NEU**
Der Standarddestruktor gibt die dynamisch erzeugten Attribute frei und speichert die drei ini-Einträge: Parallel, StateRefresh und DisplayNoActionMsg.

‣ **virtual BOOL Dlg_OnInit(HWND, HWND, LPARAM) PROTECTED NEU**
Diese Methode der Basisklasse `TModalDlg` (siehe [7]) lädt die zuletzt angezeigten Antriebe (siehe `m_Selection`) aus der ini-Datei und stellt damit die letzte Nutzeroberfläche wieder her (siehe [1], Benutzeroberfläche: / **B 10** /).

‣ **virtual void Dlg_OnCommand(HWND, int, HWND, UINT) PROTECTED NEU**
In dieser Methode der Basisklasse `TModalDlg` (siehe [7]) werden alle Steuerelement-Ereignisse des Dialogfensters verarbeitet. Die teilbereichspezifischen Botschaften werden an `TB_CheckLeft`, `TB_CheckMiddle`, `TB_CheckRight` und `TB_CheckExit` weitergeleitet. Die Steuerelemente außerhalb werden entweder direkt behandelt (‚Halbwertsbreite messen' bzw. ‚Messung abbrechen' und ‚Hilfe') oder weitergeleitet an `OnPsdOffset` (‚<PSD>-Offset') und `OnChooseTB` - CTRL+TAB zum Fokussieren des nächsten Teilbereiches unter Win32 (siehe [1], Benutzeroberfläche: / **B 120** /, / **B 130** /, / **B 140** /).

‣ **virtual void Dlg_OnHScrollBar(HWND, HWND, UINT, PROTECTED NEU**
Diese Methode der Basisklasse `TModalDlg` (siehe [7])dient zur Verarbeitung aller Ereignisse im Dialogfenster die durch Nutzereingaben an der Bildlaufleiste ausgelöst werden. Entsprechend [1] wird die Bewegung im Fahr- und Schrittbetrieb gesteuert - im Direktbetrieb wird ggf. eine Fehlermeldung ausgegeben (siehe [1], Benutzeroberfläche: / **B 70** /, / **B 80** /, / **B 100** /, / **B 110** /).

‣ **virtual BOOL LeaveDialog(void) PROTECTED NEU**
Diese Methode der Basisklasse `TModalDlg` (siehe [7]) wird beim Verlassen des Dialogfensters ausgeführt. Alle Antriebe und die Halbwertsbreitenmessung werden gestoppt.

‣ **virtual void OnTimer(const TBasicTimer*) PROTECTED NEU**
Diese Methode des Basisinterface `ITimer` (siehe [7]) wird in 100 ms Abständen aufgerufen, wenn `m_lnkTimer` gestartet ist. Hier wird die Bewegung der Antriebe überwacht - entsprechend werden `OnMotionStarts`, `OnMotionProgress` oder `OnMotionStops` bzw. `OnMeasureStarts`, `OnMeasureProgress` oder `OnMeasureStops` aufgerufen.

‣ **void OnMotionStarts(const int) PRIVATE NEU**
sucht den Teilbereich wo der Antrieb (als Index im Parameter) angezeigt wird, sperrt dort die Steuerelemente und beschriftet ‚Start' mit ‚Stop'. Außerhalb des Teilbereichs werden die Steuerelemente ‚Halbwertsbreite messen' und ‚<PSD>-Offset…' (nur wenn der PSD durch den gestarteten Antrieb bewegt werden kann) gesperrt.

‣ **void OnMotionProgress(const int) PRIVATE NEU**
sucht den Teilbereich wo der Antrieb (als Index im Parameter) angezeigt wird und aktualisiert dort alle Positionsangaben.

‣ **void OnMotionStops(const int) PRIVATE NEU**
sucht den Teilbereich wo der Antrieb (als Index im Parameter) angezeigt wird, aktualisiert dort alle Positionsangaben und beschriftet ‚Stop' wieder mit ‚Start'. Zudem werden alle beim Start gesperrten Steuerelemente (innerhalb und außerhalb der Teilbereiche) wieder freigegeben.

‣ void OnMeasureStarts(void) PRIVATE NEU
wird zum Start der Halbwertsbreitenmessung aufgerufen, sperrt die Teilbereiche sowie ‚<PSD>-Offset…', benennt ‚Halbwertsbreite messen' nach ‚Messung abbrechen' um und zeigt ‚Halbwertsbreite wird gemessen…' in der Statuszeile an.

‣ void OnMeasureProgress(void) PRIVATE NEU
liest den Fortschritt der Messung aus und zeigt ihn an.

‣ void OnMeasureStops(void) PRIVATE NEU
gibt die beim Start gesperrten Steuerelemente wieder frei und zeigt das Messergebnis oder ‚Messung wurde abgebrochen' in der Statuszeile an.

‣ void OnPsdOffset(void) PRIVATE NEU
Wenn zulässig[1], wird das Dialogfenster ‚Offset für <PSD' (siehe **IV.3**) aufgerufen (Schaltfläche ‚<PSD>-Offset…').

‣ void OnChooseTB(void) PRIVATE NEU
wird beim Drücken von [CTRL] + [TAB] aufgerufen. Wenn man sich in einem Steuerelement eines Teilbereichs befindet, wird die Antriebsauswahlliste des nächsten Teilbereichs ausgewählt. Befindet man sich außerhalb oder im letzten Teilbereich, wird der erste TB ausgewählt.

‣ void OnNoAction(void) PRIVATE NEU
Bei `m_DisplayNoActionMsg == TRUE` wird die Fehlermeldung ‚Aktion ist im aktuellen Zustand nicht verfügbar!' ausgegeben, sonst geschieht nichts.

[1] soll heißen, wenn die Funktion laut aktuellen Zustand der Antriebe und der Halbwertsbreitenmessung (siehe) zulässig ist.

- **`BOOL TB_CheckLeft( const ETB, const int )`** **PRIVATE NEU**
- **`BOOL TB_CheckMiddle( const ETB, const int )`** **PRIVATE NEU**
- **`BOOL TB_CheckRight( const ETB, const int )`** **PRIVATE NEU**

werden von der Methode `Dlg_OnCommand` aufgerufen. Im zweiten Parameter wird die Ressourcen-ID des Steuerelements übergeben, dessen Ereignis behandelt werden soll. Die Methode prüft nun für den ersten Parameter (der Teilbereich), ob die Ressourcen-ID mit einem der folgenden Steuerelemente übereinstimmt. Wenn es sich um eines der genannten Steuerelemente handelt, wird `TRUE` zurückgegeben, sonst `FALSE`.

<table>
<tr><th>Methode</th><th colspan="2">Steuerelemente</th></tr>
<tr><td>TB_CheckLeft</td><td colspan="2">Wenn zulässig, werden hier Antriebsauswahlliste, ‚Direktbetrieb', ‚Fahrbetrieb', ‚Schrittbetrieb' direkt behandelt (siehe [1], Benutzeroberfläche: / B 30 /, / B 60 /, / B 90 /).</td></tr>
<tr><td rowspan="6">TB_CheckMiddle</td><td colspan="2">Hier wird durch das Drücken von [ENTER] in den folgenden Steuerelementen zu den genannten Methoden verzweigt.</td></tr>
<tr><td>Steuerelement</td><td>Behandlungsroutine</td></tr>
<tr><td>‚Start'/ ‚Stop'</td><td>TB_StartStopClicked</td></tr>
<tr><td>‚Sollposition'</td><td>TB_AngleDestEntered</td></tr>
<tr><td>‚Geschwindigkeit'</td><td>TB_SpeedEntered</td></tr>
<tr><td>‚Schrittweite'</td><td>TB_StepWidthEntered</td></tr>
<tr><td colspan="2">(siehe [1], Benutzeroberfläche: / B 30 /, / B 60 /, / B 90 /)</td></tr>
<tr><td>TB_CheckRight</td><td colspan="2">Wenn zulässig, werden hier relative Null ‚setzen', ‚aufheben' und ‚Offset…' (Aufruf von ‚Offset für <Antrieb>' – siehe IV.2) direkt behandelt.</td></tr>
</table>

Tabelle 7 Behandelte Steuerelemente in den genannten Methoden

Durch eine Schleife über alle Teilbereiche, kann mit dem Rückgabewert `TRUE` von einer dieser Methoden geprüft werden, ob es sich um ein – mit Funktionalität belegtes – Steuerelement (ausgenommen Bildlaufleiste) handelt. Wenn zulässig wird das jeweilige Ereignis jedoch sofort behandelt!

- **`BOOL TB_CheckExit( const ETB, const int )`** **PRIVATE NEU**

wird von der Methode `Dlg_OnCommand` aufgerufen, wenn ein Steuerelement verlassen wird. Im zweiten Parameter wird die Ressourcen-ID des Steuerelements übergeben. Die Methode prüft nun für den ersten Parameter (einen Teilbereich), ob ‚Sollposition', ‚Geschwindigkeit' oder ‚Schrittweite' verlassen wurden, verweist dann zu `TB_AngleDestEntered`, `TB_SpeedEntered` bzw. `TB_StepWidthEntered` und gibt `TRUE` zurück - sonst `FALSE`.

- **`void TB_StartStopClicked( const ETB )`** **PRIVATE NEU**

Wenn zulässig, wird hier das Steuerelement ‚Start' / ‚Stop' für den im Parameter übergebenen Teilbereich behandelt (siehe [1], Benutzeroberfläche: **/ B 40 /, / B 50 /**).

- **`void TB_AngleDestEntered( const ETB, const BOOL )`** **PRIVATE NEU**

Wenn zulässig, wird hier das Steuerelement ‚Sollposition' für den im Parameter übergebenen Teilbereich behandelt. Der zweite Parameter gibt an, ob das Folgesteuerelement (hier ‚Start' / ‚Stop') ausgewählt werden soll – siehe [1].

- **`void TB_SpeedEntered( const ETB, const BOOL )`** **PRIVATE NEU**

Wenn zulässig, wird hier das Steuerelement ‚Geschwindigkeit' für den im Parameter übergebenen Teilbereich behandelt. Der zweite Parameter gibt an, ob das Folgesteuerelement (hier die Bildlaufleiste) ausgewählt werden soll – siehe [1].

- **`void TB_StepWidthEntered( const ETB, const BOOL )`** **PRIVATE NEU**

Wenn zulässig, wird hier das Steuerelement ‚Schrittweite' für den im Parameter übergebenen Teilbereich behandelt. Der zweite Parameter gibt an, ob das Folgesteuerelement (hier die Bildlaufleiste) ausgewählt werden soll – siehe [1].

- **`void TB_MotorChanged( const ETB )`** **PRIVATE NEU**

wird indirekt über die Methode `Notice` - Botschaft `USR_TB_MOTORCHANGED` aus Methode `TB_CheckLeft` - aufgerufen und behandelt. Wenn zulässig, erfolgt die Auswahl eines Antriebs in der Antriebsauswahlliste. Nur während der Halbwertsbreitenmessung wird der zuvor ausgewählte Antrieb wiederhergestellt (siehe [1], Benutzeroberfläche: / **B 20** /).

- **`void Notice( const int, const ETB, const BOOL= FALSE )`** **PRIVATE NEU**

dient zur Aktualisierung der Nutzeroberfläche im Hintergrund. Bei `m_Parallel==TRUE` werden Botschaften verschickt, die dann von Methode `Dlg_OnEvent` empfangen und quasiparallel verarbeitet werden. Bei `m_Parallel==FALSE` wird über Methode `Dlg_OnEvent` direkt zu den entsprechenden `OnTB_<irgendwas>`-Methoden verzweigt (Blockierung).

- **void OnTB_Controls(const ETB, const BOOL) PRIVATE NEU**
- **void OnTB_KindOfMove(const ETB, const BOOL) PRIVATE NEU**
- **void OnTB_StartStop(const ETB, const BOOL) PRIVATE NEU**
- **void OnTB_MotionParam(const ETB, const BOOL) PRIVATE NEU**
- **void OnTB_Positions(const ETB, const BOOL) PRIVATE NEU**
- **void OnTB_Offset(const ETB, const BOOL) PRIVATE NEU**
- **void OnTB_Units(const ETB) PRIVATE NEU**

werden von `Dlg_OnEvent` aufgerufen (siehe Methode `Notice`) und aktualisieren **Inhalt** sowie ggf. **Zustand** der genannten Steuerelemente in einem Teilbereich (erster Parameter). Wenn der zweite Parameter (falls vorhanden) `TRUE` ist, sorgen die gleichnamigen `OnTB_<irgendwas>Enable`-Methoden für die Aktualisierung des Zustands der Steuerelemente.

Methode	Steuerelemente
`OnTB_Controls`	aktualisiert **Zustand** und **Inhalt** ALLER Steuerelemente im Teilbereich.
`OnTB_KindOfMove`	wählt die ausgewählte Betriebsart (‚Direktbetrieb', ‚Fahrbetrieb', ‚Schrittbetrieb').
`OnTB_StartStop`	beschriftet ‚Start' / ‚Stop' entsprechend dem aktuellen Zustand der Antriebsbewegung.
`OnTB_MotionParam`	zeigt die zuletzt verwendeten Bewegungsparameter (‚Sollposition', ‚Geschwindigkeit', ‚Schrittweite').
`OnTB_Positions`	aktualisiert alle Positionsangaben (<Minimalposition>, ‚Istposition', <Maximalposition>, <Bildlaufleiste>, ‚Sollposition').
`OnTB_Offset`	zeigt ‚OFFSET' wenn ein Offset definiert ist, ansonsten ‚KEINER'.
`OnTB_Units`	zeigt die Einheit (**in Nutzereinheiten**) im gesamten Teilbereich.

Tabelle 8 Behandelte Steuerelemente in den genannten Methoden

- **`void OnTB_ControlsEnable( const ETB )`** **PRIVATE NEU**
- **`void OnTB_MotorListEnable( const ETB )`** **PRIVATE NEU**
- **`void OnTB_KindOfMoveEnable( const ETB )`** **PRIVATE NEU**
- **`void OnTB_StartStopEnable( const ETB )`** **PRIVATE NEU**
- **`void OnTB_MotionParamEnable( const ETB )`** **PRIVATE NEU**
- **`void OnTB_PositionsEnable( const ETB )`** **PRIVATE NEU**
- **`void OnTB_OffsetEnable( const ETB )`** **PRIVATE NEU**

werden von `Dlg_OnEvent` aufgerufen (siehe Methode `Notice`) und aktualisieren den **Zustand** der genannten Steuerelemente in einem Teilbereich (erster Parameter).

Methode	Steuerelemente
`OnTB_ControlsEnable`	aktualisiert den **Zustand** ALLER Steuerelemente im Teilbereich (auch <fehlender Referenzpunkt>).
`OnTB_MotorListEnable`	sperrt <Antrieb auswählen> während der Makroverarbeitung, ansonsten freigegeben.
`OnTB_KindOfMoveEnable`	sperrt die Betriebsarten (‚Direktbetrieb', ‚Fahrbetrieb', ‚Schrittbetrieb') oder gibt sie frei.
`OnTB_StartStopEnable`	sperrt ‚Start' / ‚Stop' oder gibt es frei.
`OnTB_MotionParamEnable`	sperrt die Bewegungsparameter (‚Sollposition', ‚Geschwindigkeit', ‚Schrittweite') oder gibt sie frei.
`OnTB_PositionsEnable`	sperrt die Positionsangaben (<Bildlaufleiste>, ‚Sollposition') oder gibt sie frei.
`OnTB_OffsetEnable`	sperrt ‚Offset…' oder gibt es frei.

Tabelle 9 Behandelte Steuerelemente in den genannten Methoden

- **`void OtherControls( void )`** **PRIVATE NEU**

wird von `Dlg_OnEvent` aufgerufen (siehe Methode `Notice`) und aktualisiert **Inhalt** und **Zustand** der Steuerelemente (‚<PSD>-Offset', ‚Halbwertsbreite messen' / ‚Messung abbrechen', <Halbwertsbreitenmessungsstatus>) außerhalb der Teilbereiche.

- **`UINT TB2Selection( const ETB )`** **PRIVATE NEU**

gibt den Index des ausgewählten Antriebs (ggf. auch `GetMotorCount` für ‚kein Antrieb') im angegebenen Teilbereich (erster Parameter) zurück.

- **`ETB Selection2TB( const UINT )`** **PRIVATE NEU**

gibt den Teilbereich zurück, indem der angebeben Index eines Antriebs ausgewählt ist (ggf. `(ETB)0`).

- **`BOOL CanSetSelection( const ETB, const UINT )`** **PRIVATE NEU**

gibt `TRUE` zurück, wenn der Index des Antriebs (zweiter Parameter) im Teilbereich (der erster Parameter angegeben wird) ausgewählt werden kann. Wenn der Antrieb bereits in einem anderen Teilbereich angezeigt wird, wird `FALSE` zurückgegeben.

- **`BOOL SetSelection( const ETB, const UINT )`** **PRIVATE NEU**

Wenn der Antrieb ausgewählt werden darf (`CanSetSelection`) wird er ausgewählt. Die Einstellung wird gespeichert und die zuletzt verwendeten Daten werden angezeigt (`OnTB_Controls` und `OnTB_Units`), `TRUE` wird zurückgebeben. Wenn der Antrieb bereits in einem anderen Teilbereich angezeigt wird, geschieht nichts und es wird `FALSE` zurückgegeben.

HWND GetCtrl(const ETB, const ERessourceId) PRIVATE NEU
gibt das Handle auf dein Steuerelement zurück. Der erste Parameter ist der Teilbereich, der Zweiten ist zur Adressierung des Steuerelements - ggf. `0`. Die Methode verwendet `GetResId`.

int GetResId(const ETB, const ERessourceId) STATIC PRIVATE NEU
gibt die Ressourcen-ID des im ersten (Teilbereich) und zweiten Parameter (Adressierung eines Elements) angegebenen Steuerelements zurück - ggf. `-1` (siehe **Tabelle 6**).

BOOL TB_Next(ETB &) STATIC PRIVATE NEU
gibt per Referenz den Nachfolger des, im ersten Parameter angebenen, Teilbereichs zurück. Wenn `(ETB)0` angegeben wird, wird der erste Teilbereich zurückgeben. Wird der letzte Teilbereich angegeben, wird dieser zurückgegeben und der Rückgabewert wird `FALSE` (d.h. keine weiteren Teilbereiche). Mit

```
ETB tb= (ETB)0;
while (TB_Next(tb) ) /* place action here*/;
```

kann über alle Teilbereiche iteriert werden.

UINT TB_Count(void) STATIC PRIVATE NEU
gibt die Anzahl der Teilbereiche zurück und verwendet `TB_Next`.

UINT TB_GetId(const ETB) STATIC PRIVATE NEU
gibt eine fortlaufende Nummer für den angebenen Teilbereich zurück. Der erste Teilbereich entspricht `0`. Wird für `TB_GetName` benötigt.

void TB_GetName(const ETB aTB, LPSTR aReturn) STATIC PRIVATE NEU
gibt den Namen der ini-Sektion im zweiten Parameter zurück. Dort muss genügend Platz durch den Aufrufer reserviert sein.

void ScrollbarCalculate(const TAngle, const TAngle, const TAngle, int&, int&, int&) STATIC PRIVATE NEU
rechnet die Antriebspositionen (erster Parameter = Minimalposition, Zweiter = Istposition, Dritter = Maximalposition) in die ganzzahligen Angaben für die Bildlaufleiste (Vierter = Minimalposition, Fünfter = Position des Bildlauffeldes, Sechster = Maximalposition) um.

IV.1.4 Ressourcen

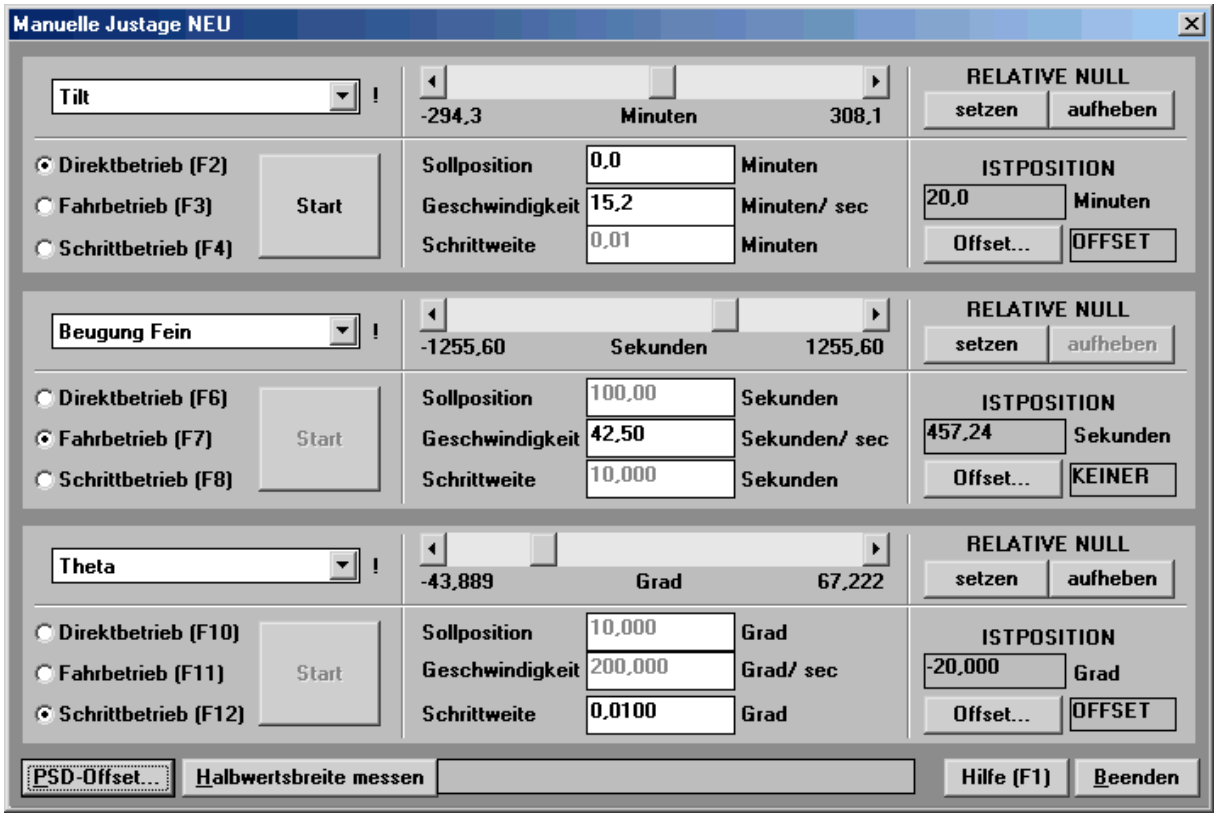

Abbildung 4 Hauptdialog neue Manuelle Justage (Quelle: [9])

Ressourcen-ID	Element	Bezeichnung
cm_p1_Offset	Schaltfläche	‚Offset…'
cm_p1_ResetRelativeZero	…	‚aufheben'
cm_p1_SetRelativeZero	…	‚setzen'
cm_p1_Start	…	‚Start'
id_p1_MotorList	Auswahlliste	Antriebsliste
id_p1_DirectMode	Optionsfeld	‚Direktbetrieb'
id_p1_DriveMode	…	‚Fahrbetrieb'
id_p1_StepMode	…	‚Schrittbetrieb'
id_p1_Angle	Eingabefeld	Istposition
id_p1_AngleDest	…	Sollposition
id_p1_OffsetState	…	Offset-Status
id_p1_Speed	…	Geschwindigkeit
id_p1_StepWidth	…	Schrittweite
id_p1_RelativePos	Bildlaufleiste	relative Position
txt_p1_AngleDestUnit	Textfeld	Einheit für Sollposition
txt_p1_AngleUnit	…	Einheit für Istposition
txt_p1_RelativePosMax	…	Maximalposition
txt_p1_RelativePosMin	…	Minimalposition
txt_p1_RelativPosUnit	…	Einheit für relative Position
txt_p1_SpeedType	…	Einheit für Geschwindigkeit
txt_p1_StepWidthType	…	Einheit für Schrittweite
txt_p1_RefPoint	…	Kennzeichnung für fehlenden Referenz-punktlauf

Tabelle 10 Ressourcen im Teilbereich des Hauptdialogfensters – „p1" steht für 1., „p2" für 2. und „p3" für 3. Teilbereich

Ressourcen-ID	Element	Bezeichnung
ManualAdjustmentDlg	Dialogfenster	‚Manuelle Justage'
ManualAdjustmentDlgEng	...	‚Manual Adjustment'
IDABORT	Schaltfläche	‚Beenden'
cm_HwbMeasure	…	‚Halbwertsbreite messen'
cm_PsdOffset	…	‚PSD-Offset…'
id_HwbState	Eingabefeld	Status der Messung

Tabelle 11 Ressourcen für die weiteren Elemente des Hauptdialogs

IV.1.5 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚Lines Of Code' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	1.249
	‚LOC of Implementation'*	LOCI	1.064
	‚LOC of Declaration'*	LOCD	185
	‚Number Of Attributes'	NOA (0, 30)	6
	‚Number Of Operations'	NOO (0, 50)	**52**
	‚Number Of Members' Attribute + Methoden	NOM = NOA + NOO	58
	‚Number Of Constructors'	NOCON (0, 5)	1
	‚Number Of Overridden Methods'	NOOM (0, 10)	6
	‚Percentage of Private Members'	PPrivM	87
	‚Percentage of Protected Members'	PProtM (0, 10)	10
	‚Percentage of Public Members'	PPubM	3
	‚Weighted Methods Per Class'	WMPC1 (0, 30)	
Attribute			
	‚Attribute Complexity'	AC	54
Methoden			
	‚Maximum Number Of Parameters'	MNOP (0, 4)	**6**
	‚Cyclomatic Complexity'	CC	
Kommentare			
	‚Number Of Comments'*	NOC	412
	‚True Comment Ratio'	TCR (5, 400)	47

Tabelle 12 Ausgewählte Metriken der Klasse `TManJustageDlg` (Quelle: Together ‚Version 5.5)

* Diese Metriken sind nicht Bestandteil von Together, sondern wurden manuell ermittelt.

IV.2 Klasse TMotorOffsetDlg

Deklaration : MJ_GUI.H

Implementation: MJ_GUI.CPP

Die Klasse `TMotorOffsetDlg` ist von der Basisklasse für modale Dialogfenster `TModalDlg` abgeleitet.

IV.2.1 Attribute

`TManJustage *m_lnkManJustage` **PRIVATE NEU**

ist die Referenz auf die Funktionalität (wird im Konstruktor vom Aufrufer übergeben) und darf nicht im Destruktor freigegeben werden.

`UINT m_Idx` **PRIVATE NEU**

gibt an, für welchen Antrieb (als Index in `lpMLIST`) das Offset definiert werden soll (wird zur Steuerung der Funktionalität benötigt) – wird im Konstruktor vom Aufrufer übergeben.

`UINT m_Digits` **PRIVATE NEU**

Ist die Nachkommastellengenauigkeit für Positionsangaben des gewählten Antriebs (im Konstruktor initialisiert).

IV.2.2 Methoden

▸ `TMotorOffsetDlg( const UINT, TManJustage* )` PUBLIC NEU

Dieser Konstruktor dient zur Vorbereitung der Anzeige des dazugehörigen Dialogfensters. Dazu wird der Konstruktor der Basisklasse `TModalDlg` (siehe [7]) gerufen.

Der erste Parameter gibt den Index des Antriebs in `lpMList` (wird in `m_Idx` gespeichert) und der Zweite eine Referenz auf die Funktionalität (gespeichert in `m_lnkManJustage`) an. Initialisiert wird auch `m_Digits`.

▸ `BOOL Dlg_OnInit( HWND, HWND, LPARAM )` PROTECTED NEU

ist eine Methode der Basisklasse `TModalDlg` (siehe [7]) und bereitet das Dialogfenster für das Betreten vor. Dazu werden alle Eingabe- und Textfelder mit entsprechenden/ aktuellen Werten belegt.

▸ `void Dlg_OnCommand( HWND, int, HWND, UINT )` PROTECTED NEU

Ist eine Methode der Basisklasse `TModalDlg` (siehe [7]). Hier werden alle Steuerelement-Ereignisse des Dialogfensters verarbeitet. Bei der Auswahl eines Optionsfeldes wird zu Methode `ChooseAngleMode` verwiesen, wobei der Parameter angibt, dass ‚entspricht Winkel' ausgewählt wurde. Bei Eingaben oder beim Verlassen des Dialogfensters wird zu Methode `CalcValue` verwiesen (wenn der Parameter `TRUE` ist, werden die Werte in die Funktionalität übernommen).

▸ `void ChooseAngleMode( BOOL )` PRIVATE NEU

Entsprechend dem Parameter wird ‚entspricht Winkel' oder ‚Offset eingeben' ausgewählt und das zugehörige Eingabefeld wird freigegeben oder gesperrt.

▸ `BOOL CalcValue( BOOL )` PRIVATE NEU

Berechnet, in Abhängigkeit ob ‚entspricht Winkel' oder ‚Offset eingeben' ausgewählt ist, ein neues Offset für den Antrieb. Wenn der Parameter `TRUE` angibt, wird das Berechnete in der Funktionalität gespeichert. Bei `FALSE` werden die berechneten Werte nur im Dialogfenster angezeigt.

Der Rückgabewert gibt an, ob alle erforderlichen Benutzereingaben vollständig und korrekt waren. Bei `FALSE` konnte keine Berechnung durchgeführt werden.

▸ `BOOL IsAngleMode( void )` PRIVATE NEU

Gibt zurück, ob ‚entspricht Winkel' (`TRUE`) oder ‚Offset eingeben' (`FALSE`) ausgewählt ist.

IV.2.3 Ressourcen

Abbildung 5 Offset für <Antrieb> (Quelle: [9])

Ressourcen-ID	Element	Bezeichnung
MotorOffsetDlg	Dialogfenster	‚Offset für <Antrieb>'
MotorOffsetDlgEng	…	‚Offset for <Antrieb>'
IDCANCEL	Schaltfläche	‚Abbruch'
IDOK	…	‚OK'
id_AngleMode	Optionsfeld	‚entspricht Winkel'
id_OffsetMode	…	‚Offset eingeben'
id_Offset	Eingabefeld	‚aktueller Offset'
id_Angle	…	‚aktuelle Istposition'
id_OffsetDest	…	‚Offset eingeben'
id_AngleDest	…	‚entspricht Winkel'
txt_OffsetUint	Textfeld	Einheit für ‚aktueller Offset'
txt_AngleUnit	…	Einheit für ‚aktuelle Istposition'
txt_OffsetDestUnit	…	Einheit für ‚Offset eingeben'
txt_AngleDestUint	…	Einheit für ‚entspricht Winkel'

Tabelle 13 Ressourcen für den ‚Offset für <Antrieb>'

IV.2.4 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚**L**ines **O**f **C**ode' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	140
	‚**LOC** of **I**mplementation'*	LOCI	113
	‚**LOC** of **D**eclaration'*	LOCD	27
	‚**N**umber **O**f **A**ttributes'	NOA (0, 30)	3
	‚**N**umber **O**f **O**perations'	NOO (0, 50)	5
	‚**N**umber **O**f **M**embers' Attribute + Methoden	NOM = NOA + NOO	8
	‚**N**umber **O**f **C**onstructors'	NOCON (0, 5)	1
	‚**N**umber **O**f **O**verridden Methods'	NOOM (0, 10)	2
	‚**P**ercentage of **Priv**ate **M**embers'	PPrivM	67
	‚**P**ercentage of **Prot**ected **M**embers'	PProtM (0, 10)	**22**
	‚**P**ercentage of **Pub**lic **M**embers'	PPubM	11
	‚**W**eighted **M**ethods **P**er **C**lass'	WMPC1 (0, 30)	
Attribute			
	‚**A**ttribute **C**omplexity'	AC	27
Methoden			
	‚**M**aximum **N**umber **O**f **P**arameters'	MNOP (0, 4)	4
	‚**C**yclomatic **C**omplexity'	CC	
Kommentare			
	‚**N**umber **O**f **C**omments'*	NOC	55
	‚**T**rue **C**omment **R**atio'	TCR (5, 400)	45

Tabelle 14 Ausgewählte Metriken der Klasse TMotorOffsetDlg (Quelle: Together ‚Version 5.5)

* Diese Metriken sind nicht Bestandteil von Together, sondern wurden manuell ermittelt.

IV.3 Klasse TPsdOffsetDlg

Deklaration : MJ_GUI.H

Implementation: MJ_GUI.CPP

Die Klasse `TPsdOffsetDlg` ist von der Basisklasse für modale Dialogfenster `TModalDlg` abgeleitet.

IV.3.1 Attribute

`TManJustage *m_lnkManJustage` **PRIVATE NEU**

Ist eine Referenz auf die Funktionalität (wird im Konstruktor vom Aufrufer übergeben) und darf nicht im Destruktor freigegeben werden.

`UINT m_Idx` **PRIVATE NEU**

gibt an, für welchen Antrieb (als Index in `lpMLIST`) das Offset definiert werden soll (wird zur Steuerung der Funktionalität benötigt) – wird im Konstruktor vom Aufrufer übergeben.

`UINT m_Digits` **PRIVATE NEU**

ist die Nachkommastellengenauigkeit für Positionsangaben des gewählten Antriebs (im Konstruktor initialisiert).

IV.3.2 Methoden

`TPsdOffsetDlg( const UINT, TManJustage* )` **PUBLIC NEU**

Dieser Konstruktor dient zur Vorbereitung der Anzeige des dazugehörigen Dialogfensters. Dazu wird der Konstruktor der Basisklasse `TModalDlg` (siehe [7]) gerufen.

Der erste Parameter gibt den Index des Antriebs in `lpMList` (wird in `m_Idx` gespeichert) und der Zweite eine Referenz auf die Funktionalität (gespeichert in `m_lnkManJustage`) an. Initialisiert wird auch `m_Digits`.

`BOOL Dlg_OnInit( HWND, HWND, LPARAM )` **PROTECTED NEU**

ist eine Methode der Basisklasse `TModalDlg` (siehe [7]) und bereitet das Dialogfenster für das Betreten vor. Dazu werden alle Eingabe- und Textfelder mit entsprechenden/ aktuellen Werten belegt.

`void Dlg_OnCommand( HWND, int, HWND, UINT )` **PROTECTED NEU**

ist eine Methode der Basisklasse `TModalDlg` (siehe [7]). Hier werden alle Steuerelement-Ereignisse des Dialogfensters verarbeitet. Bei Eingaben oder beim Verlassen des Dialogfensters wird zu Methode `CalcValue` verwiesen (wenn der Parameter `TRUE` ist, werden die Werte in die Funktionalität übernommen).

`BOOL CalcValue( BOOL )` **PRIVATE NEU**

Berechnet, in Abhängigkeit ob ‚entspricht Winkel' oder ‚Offset eingeben' ausgewählt ist, ein neues Offset für den Antrieb. Wenn der Parameter `TRUE` angibt, wird das berechnete in der Funktionalität gespeichert. Bei `FALSE` werden die berechneten Werte nur im Dialogfenster angezeigt.

Der Rückgabewert gibt an, ob alle erforderlichen Benutzereingaben vollständig und korrekt waren. Bei `FALSE` konnte keine Berechnung durchgeführt werden.

IV.3.3 Ressourcen

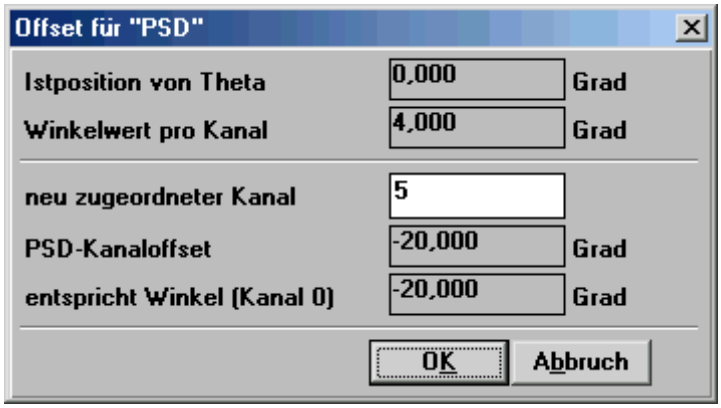

Abbildung 6 Offset für <PSD> (Quelle: [9])

Ressourcen-ID	Element	Bezeichnung
PsdOffsetDlg	Dialogfenster	‚Offset für <PSD>'
PsdOffsetDlgEng	…	‚Offset for <PSD>'
IDCANCEL	Schaltfläche	‚Abbruch'
IDOK	…	‚OK'
id_ChannelDest	Eingabefeld	‚neu zugeordneter Kanal'
id_PsdAngle		‚Istposition von <Theta>'
id_PsdAngleDest	…	‚Winkelwert pro Kanal'
id_PsdAngleDest	…	‚entspricht Winkel (Kanal 0)'
id_PsdOffsetDest		‚<PSD>-Kanaloffset'
txt_PsdAngleDestUint	Textfeld	Einheit für ‚Winkelwert pro Kanal'
txt_PsdAngleDestUnit	…	Einheit für ‚entspricht Winkel (Kanal 0)'
txt_PsdAngleDestUnit	…	‚Istposition für <Theta>'
txt_PsdAngleUnit	…	Einheit für ‚Istposition von <Theta>'
txt_PsdOffsetDest	…	‚<PSD>-Kanaloffset'
txt_PsdOffsetDestUnit	…	Einheit für ‚<PSD>-Kanaloffset'

Tabelle 15 Ressourcen für ‚Offset für <PSD>'

IV.3.4 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚**L**ines **O**f **C**ode' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	118
	‚**LOC** of **I**mplementation'*	LOCI	90
	‚**LOC** of **D**eclaration'*	LOCD	28
	‚**N**umber **O**f **A**ttributes'	NOA (0, 30)	3
	‚**N**umber **O**f **O**perations'	NOO (0, 50)	3
	‚**N**umber **O**f **M**embers' Attribute + Methoden	NOM = NOA + NOO	6
	‚**N**umber **O**f **C**onstructors'	NOCON (0, 5)	1
	‚**N**umber **O**f **O**verridden Methods'	NOOM (0, 10)	2
	‚**P**ercentage of **Priv**ate **M**embers'	PPrivM	57
	‚**P**ercentage of **Prot**ected **M**embers'	PProtM (0, 10)	**29**
	‚**P**ercentage of **Pub**lic **M**embers'	PPubM	14
	‚**W**eighted **M**ethods **P**er **C**lass'	WMPC1 (0, 30)	
Attribute			
	‚**A**ttribute **C**omplexity'	AC	27
Methoden			
	‚**M**aximum **N**umber **O**f **P**arameters'	MNOP (0, 4)	4
	‚**C**yclomatic **C**omplexity'	CC	
Kommentare			
	‚**N**umber **O**f **C**omments'*	NOC	49
	‚**T**rue **C**omment **R**atio'	TCR (5, 400)	53

Tabelle 16 Ausgewählte Metriken der Klasse `TPsdOffsetDlg` (Quelle: Together ‚Version 5.5)

* Diese Metriken sind nicht Bestandteil von Together, sondern wurden manuell ermittelt.

V Attribute

- **`extern TSteering Steering`** **GLOBAL NEU**

ist ein Objekt für den Zugriff auf die Makrosteuerung.

- **`extern LPMList lpMList`** **GLOBAL NEU**

ist eine Komposition aus `TMotor`-Objekten zur Verwaltung der angeschlossenen Antriebe.

Anhang A – VERWANDTE DOKUMENTE

[1] „Pflichtenheft ‚Manuelle Justage' ", Version 2.1 von Thomas Kullmann und Günther Reinecker

[2] Bewertung der Neuentwürfe des Oberflächenfensters zur Manuellen Justage", Version 1.7 von Thomas Kullmann und Günther Reinecker

[3] „semantische Fehler in alten Dialogfenster ‚Manuellen Justage'", Version 1.0 von Thomas Kullmann und Günther Reinecker

[4] „Lehrbuch der Software-Technik: Software Management, Software Qualitätssicherung, Unternehmensmodellierung ", Berlin: Spektrum, Akad. Verl., 1998, von Helmut Balzert

[5] „Reverse-Engineering der objektorientierten Teile des Subsystems Motorsteuerung", Version 1.3 von Thomas Kullmann und Günther Reinecker

[6] „Reverse-Engineering des Subsystems Ablaufsteuerung", Version 1.0 von Thomas Kullmann und Günther Reinecker

[7] „Reverse-Engineering der Basisklassen für Dialogfenster", Version 1.2 von Thomas Kullmann und Günther Reinecker

[8] „Reverse-Engineering des Subsystems Detektoren des RTK-Steuerprogramms", November 2000 von Jan Picard, René Harder und Alexander Paschold

[9] Human-Interface Prototypen *Manuelle Justage*, von Thomas Kullmann und Günther Reinecker

[10] „Layoutkonventionen und Steuerelemente", Version 1.0 von Thomas Kullmann und Günther Reinecker

Anhang B – TABELLEN

Anhang C – ABBILDUNGEN