

RE-ENGINEERING DES SUBSYSTEMS ABLAUFSTEUERUNG

Dokument zur Diplomarbeit
- Designphase -

Autoren	Thomas Kullmann, Günther Reinecker
Dokumentversion	1.0
Zustand	abgeschlossen
letzte Bearbeitung	13.08.02



Inhalt

I	ÜBERBLICK	3
II	TYPEN.....	5
III	KONSTANTEN	9
IV	KLASSEN	10
IV.1	Klasse TSteering.....	11
	IV.1.1 Friends	11
	IV.1.2 Attribute	11
	IV.1.3 Methoden.....	16
	IV.1.4 Bewertung	23
IV.2	Klasse TCmd.....	24
	IV.2.1 Friends	24
	IV.2.2 Attribute	24
	IV.2.3 Methoden.....	25
	IV.2.4 Bewertung	27
IV.3	Klasse TChooseAxisCmd.....	28
	IV.3.1 Methoden.....	28
	IV.3.2 Bewertung	28
IV.4	Klasse TSetWidthCmd.....	28
	IV.4.1 Methoden.....	29
	IV.4.2 Bewertung	29
IV.5	Klasse TLoadPointCmd.....	29
	IV.5.1 Methoden.....	30
	IV.5.2 Bewertung	30
IV.6	Klasse TMoveToPointCmd.....	30
	IV.6.1 Methoden.....	31
	IV.6.2 Bewertung	31
IV.7	Klasse TChooseDetectorCmd.....	32
	IV.7.1 Methoden.....	32
	IV.7.2 Bewertung	32
IV.8	Klasse TGotoIntensityCmd	33
	IV.8.1 Attribute	33
	IV.8.2 Methoden.....	34
	IV.8.3 Bewertung	36
IV.9	Klasse TGotoPeakCmd.....	36
	IV.9.1 Attribute	37
	IV.9.2 Methoden.....	38
	IV.9.3 Bewertung	40
IV.10	Klasse TShowValueCmd	41
	IV.10.1 Methoden.....	41
	IV.10.2 Bewertung	41
IV.11	Klasse TCalculateCmd.....	42
	IV.11.1 Methoden.....	42
	IV.11.2 Bewertung	43
IV.12	Klasse TInquireCmd.....	43
	IV.12.1 Methoden.....	43
	IV.12.2 Bewertung	44
IV.13	Klasse TControlFlankCmd.....	44
	IV.13.1 Attribute	44
	IV.13.2 Methoden.....	45
	IV.13.3 Bewertung	46
IV.14	Klasse TSetupScanCmd.....	46
	IV.14.1 Attribute	46
	IV.14.2 Methoden.....	46
	IV.14.3 Bewertung	47



IV.15 Klasse TSetFileNameCmd	47
IV.15.1 Attribute	47
IV.15.2 Methoden.....	47
IV.15.3 Bewertung	48
IV.16 Klasse TSaveDataCmd.....	48
IV.16.1 Attribute	48
IV.16.2 Methoden.....	48
IV.16.3 Bewertung	49
IV.17 Klasse TScanCmd.....	49
IV.17.1 Attribute, Methoden	49
IV.17.2 Bewertung	50
IV.18 Klasse TAreaScanCmd	50
IV.18.1 Attribute, Methoden	50
IV.18.2 Bewertung	51
V ATTRIBUTE.....	51
VI METHODEN	53
VII ANHANG	53
VII.1 Verwandte Dokumente	53
VII.2 Index.....	53
VII.3 Tabellen.....	54
VII.4 Abbildungen.....	54

I Überblick

Im Mittelpunkt bei der Erarbeitung dieses Dokuments stand die Vollständigkeit. Jeder Typ, jede Klasse, jeder Member, u.ä. wird kurz und präzise erläutert. Dazu wurde dieses Dokument stark formalisiert, die Layoutkonventionen sind unter [7] zu finden. Weiterführende Informationen zum Subsystem Antriebssteuerung findet man unter [1] und [2]. In [3] kann zum Thema Detektoren nachgeschlagen werden.

Um das Dokument übersichtlicher zu gestalten, wurden die behandelten Attribute und Methode nach "Sinn-einheiten" geordnet.

► Dokumentation des Istzustands und Änderungen am Subsystem

Die Dokumentation bezieht sich auf den Quellcode der hier aufgelisteten Dateien (Stand: 05.08.2002). Nachfolgende Änderungen oder Neuimplementierungen können nicht berücksichtigt werden.

h-Dateien (Deklaration)	cpp-Dateien (Implementation)
<ul style="list-style-type: none"> • M_STEERG.H • WORKFLOW.H 	<ul style="list-style-type: none"> • M_STEERG.CPP

Tabelle 1 „Auflistung der zum Subsystem zugehörigen Dateien“ (Quelle: selbst)

Im Gegensatz zu Version 1.9 werden hier auch die Änderungen am Subsystem dokumentiert und erklärt. Änderungen wurde **rot** (nicht fett und damit im Gegensatz zu Änderungswünschen, Fehlerquellen, etc.) hervorgehoben. Neue Member sind durch **NEU**, geänderte Signaturen sind durch **GEÄ** und auskommentierte Strukturen sind mit **ENTF** in der letzten Spalte (nach dem Zugriffsschutz, z.B. **PRIVATE**) markiert.

Die deutlich gestiegene LOC-Anzahl ist auf die neue ausführliche Kommentierung und die zahlreich eingefügten Leerzeilen, die die Lesbarkeit in großen Methoden verbessern sollen, zurückzuführen. Kaum Einfluss haben die neuen Member, weil sie fast ausschließlich aus dem globalen Namensraum in TSteering verschoben wurden. Der Fährnis halber wurde der auskommentierte, tote Code entfernt, um die Metriken nicht zu verfälschen.

Die Ablaufsteuerung wird über das `Steering`-Objekt, das in Datei `M_STEERG.CPP` deklariert ist, reali-



siert. Sie besteht aus einer Reihe von **Makros**¹, die ihrerseits jeweils aus einer “nahezu beliebige“ Kombination von vordefinierten **Kommandos**² bestehen können. Diese können wiederum durch die Angabe von **Kommandoparametern** individualisiert werden und besitzen eine gewisse Einzelfunktionalität. Aus der Reihenfolge/ dem Ablauf der Kommandos ergibt sich für das Makro eine Gesamtfunktionalität, die beim Ausführen des Makros (und somit dem Ablauf der einzelnen Kommandos im Makro) durchgeführt wird. Die Verarbeitung des Makros kann zwischen den Kommandos und den **Einzelschritten**, aus dem die Kommandos bestehen, gestoppt und zu einem späteren Zeitpunkt wieder fortgesetzt werden. Die Makroverarbeitung kann jedoch auch abgebrochen werden, so dass ein Fortsetzen unmöglich ist.

TSteering enthält in aMacroList eine Liste der geladenen Makros. Derzeit ist ihre Größe (und damit auch die maximale Anzahl von Makros) statisch auf 20 begrenzt, kann aber unproblematisch erhöht werden (siehe Anmerkungen bei TMacroTag). Mit der Methode Initialize können Makros aus STANDARD.MAK und mit LoadMacroByUser aus SCAN.MAK geladen werden. Die Syntax der mak-Dateien hat Kay Schützler unter [4] zusammengestellt, gültige Parameterbelegungen der verwendeten Kommandos sind beim jeweiligen Kommando selbst, beim Konstruktor aufgezählt – hier wird auch deren Funktion erklärt. Nur bei sehr komplexen Kommandos, die aus mehreren Einzelschritten bestehen, war es vorteilhaft, die Beschreibung der Funktionalität und der Parameterbelegung direkt bei der Klasse zu erklären, nicht im Konstruktor.

Jedes Makro wird durch eine Struktur vom Typ TMacroTag repräsentiert. Entsprechend dem Namen, in der mak-Datei, wird jedem Makro ein eindeutiger Typ (TMacroId) zugeordnet, auch dieser behindert das Hinzufügen von Makros, weil nur maximal ein Makro eines Typs in aMacroList vorhanden sein kann (bereits vorhandene Makros werden ggf. überschrieben). Neben Typ, dem Namen des Makros und der Datei, aus der das Makro geladen wurde, enthält TMacroTag auch eine dynamische Liste von TCmdTag-Objekten, die die Kommandos repräsentieren – sie kann nahezu beliebig erweitert werden. Die Liste der möglichen Kommandos ist in **Tabelle 4** dargestellt.

Während der Abarbeitung eines Makros wird für das aktuelle Kommando meist ein Objekt, das von TCmd abgeleitet ist, erstellt, das die gewünschte Funktionalität durchführt. Nur für nicht mehr verwendete Kommandos und Kontrollfluss-Kommandos, die die Abarbeitung des Makros beeinflussen³, wird kein Objekt erstellt – ihre Funktionalität wurde direkt in TSteering, Methode StartCmdExecution implementiert, wo das nächste auszuführende Kommando ermittelt wird.

¹ eine Art vereinfachtes Programm

² ausgewählten Befehlen

³ bedingte und unbedingte Sprünge, sowie Stop



II Typen

► `typedef enum { ... } TMacroId`

ordnet jedem Makro einen eindeutigen Typ zu – im Gegensatz zur “beliebigen“, textuellen Bezeichnung der mak-Datei (Bezeichnungen sind identisch zu den verwendeten, symbolischen Werten)

“beliebige“ Bezeichnung	TMacroId	Verwendung
“SetupTopography“	SetupTopography	Einstellen des Arbeitspunkt STANDARD.MAK: 4 Kommandos bei TTopographyExecute benutzt
“SearchReflection“	SearchReflection	Peak-Suche STANDARD.MAK: 5 Kommandos bei TAdjustmentExecute benutzt
“InquireHwb“	InquireHwb	Bestimmen der Halbwertsbreite STANDARD.MAK: 16 Kommandos bei TAdjustmentExecute und TAngleControl benutzt
“AzimutalJustify“	AzimutalJustify	Makro zur Azimutalen Justage STANDARD.MAK: 22 Kommandos bei TAdjustmentExecute benutzt
“Test“	Test	Test STANDARD.MAK: 4 Kommandos bei TAdjustmentExecute benutzt
“ScanJob“	ScanJob	Standard zur automatischen Ausführung von Scan's SCAN.MAK: 30 Kommandos nie ausgeführt
“AreaScanJob“	AreaScanJob	Standard zur automatischen Ausführung von AreaScan's SCAN.MAK: 10 Kommandos nie ausgeführt
“BatchMacro“	BatchMacro	nicht verwendet; AUSKOMMENTIERT
“MiddleOfValley“	MiddleOfValley	nicht verwendet; AUSKOMMENTIERT

Tabelle 2 „Beziehungen zwischen Bezeichnung und eindeutigem Makrotyp – Groß-/ Kleinschreibung ist zu beachten!“
(Quelle: nach STANDARD.MAK und SCAN.MAK)

Der Typ wird aus der “beliebigen“, textuellen Bezeichnung (die im jeweils bei [COMMON] -> Name in der mak-Datei steht) ermittelt. Die Bezeichnung wird bei Methode TSteering::ParsingMacroId in den Typ übersetzt. Eine Umkehrfunktion ist nicht erforderlich.

**► typedef struct { ... } TMacroTag**

repräsentiert ein Makro; besteht aus dessen Typ, Namen, mak-Datei und der Kommandoliste

Attribut in TCmdTag		Verwendung
► TMacroId Id	PUBLIC	eindeutiger Typ des Makros
► BOOL bIsReady	PUBLIC	dieses Flag kennzeichnet, das das Makro erfolgreich eingelesen wurde wird nie auf FALSE gesetzt; Redundant, weil fehlerhafte Makros nicht in TSteering::aMacroList aufgenommen werden
► char Name [_MAXLENMACRONAME+1]	PUBLIC	“beliebige“, textuelle Bezeichnung des Makros, so wie diese in Dialogfenstern anzuzeigen ist; Für die korrekte Initialisierung von Id ist es erforderlich, dass nur Werte aus Tabelle 2 verwendet werden
► char FileName [_MAXLENMAKFILENAME+1]	PUBLIC	enthält Dateinamen und Erweiterung der mak-Datei, aus der das Makro geladen wurde (derzeit nur STANDARD.MAK oder SCAN.MAK)
► TCmdTag* CmdList	PUBLIC	Zeiger auf die Kommandoliste
► int Length	PUBLIC	Anzahl der Kommandos in CmdList

Tabelle 3 „Attribute von TMacroTag“ (Quelle: selbst)



► **typedef enum { ... } TCmdId**

ordnet jedem Kommando einen eindeutigen Typ zu – im Gegensatz zur “beliebigen“, textuellen Bezeichnung in der mak-Datei (Bezeichnungen sind identisch zu den verwendeten, symbolischen Werten)

“beliebige“ Bezeichnung	TCmdId	Klasse für die gewünschte Funktionalität
"ChooseAxis"	ChooseAxis	TChooseAxisCmd
"SetWidth"	SetWidth	TSetWidthCmd
"LoadPoint"	LoadPoint	TLoadPointCmd
"MoveToPoint"	MoveToPoint	TMoveToPointCmd
"ChooseDetector", "ChooseDevice"	ChooseDetector	TChooseDetectorCmd
"GotoIntensity"	GotoIntensity	TGotoIntensityCmd
"GotoPeak "	GotoPeak	TGotoPeakCmd
"ShowValue"	ShowValue	TShowValueCmd
"Calculate"	Calculate	TCalculateCmd
"ControlFlank"	ControlFlank	TControlFlankCmd
"SetupScan"	SetupScan	TSetupScanCmd
"SetFileName"	SetFileName	TSetFileNameCmd
"SaveData"	SaveData	TSaveDataCmd
"Scan"	Scan	TScanCmd
"AreaScan"	AreaScan	TAreaScanCmd
"Inquire"	Inquire	<u>bedingter Sprung</u> bei InquireResult == TRUE wird das nächste (sonst das übernächste) Kommando im Makro ausgeführt ⁴
"GotoLine"	GotoLine	<u>unbedingter Sprung</u> es wird das (als erster <i>Kommandoparameter</i> angegebene) Kommando ausgeführt
"Stop"	Stop	kennzeichnet das <u>Ende einer Kommandofolge</u> in einem Makro; ggf. wird dieses Kommando beim Einlesen des Makros ergänzt
"GetHWB"	GetHWB	nicht verwendet; AUSKOMMENTIERT
"ShowHWB"	ShowHWB	nicht verwendet; AUSKOMMENTIERT
"SetupAreaScan"	SetupAreaScan	nur bei Methode TSteering::ParsingCmd verwendet; AUSKOMMENTIERT

Tabelle 4 „Beziehungen zwischen Bezeichnung, eindeutigem Kommandotyp und Benutzung – Groß-/ Kleinschreibung ist zu beachten!“ (Quelle: selbst)

Die Methode TSteering::ParsingCmd ist für die Abbildung von *Bezeichnung* auf TCmdId zuständig. Eine Umkehrfunktion ist nicht erforderlich.

⁴ Wenn dabei das letzte Kommando erreicht wird oder eine Position außerhalb der Kommandoliste angesprungen werden würde, ist die Makrobearbeitung beendet.



► **typedef enum { ... } TiParam**

ordnet "ausgewählten" *Kommandoparametern* einen eindeutigen Typ zu – im Gegensatz zur "beliebigen", textuellen Bezeichnung in der mak-Datei (Bezeichnungen sind identisch zu den verwendeten, symbolischen Werten); Solche Parameter dienen zur näheren Präzisierung des Kommandos selbst oder sie geben an, wie nachfolgenden Parameter (z.B. Positionsangaben) zu interpretieren sind.

"beliebige" Bezeichnung	TiParam	"beliebige" Bezeichnung	TiParam
"AbsorberUsed"	AbsorberUsed	"MaximizeCollimator"	MaximizeCollimator
"AreaScanResult"	AreaScanResult	"MaximizeGradient"	MaximizeGradient
"Argument"	Argument	"MaximizeTilt"	MaximizeTilt
"Array"	Array	"Middle"	Middle
"BackMove"	BackMove	"Min"	Min
"DecreaseWidth"	DecreaseWidth	"Opposite"	Opposite
"Difference"	Difference	"Peak"	Peak
"DynamicWidth"	DynamicWidth	"Reflection"	Reflection
"Equidistant"	Equidistant	"Relative"	Relative
"ForAreaScan"	ForAreaScan	"Result"	Result
"ForScan"	ForScan	"ScanResult"	ScanResult
"Hwb"	Hwb	"SmallSide"	SmallSide
"IncreasePeak"	IncreasePeak	"Standard"	Standard
"Interpolation"	Interpolation	"Start"	Start
"LargeSide"	LargeSide	"StaticStepWidth"	StaticStepWidth
"LastGoal"	LastGoal	"ThisDFPos"	ThisDFPos
"List"	List	"ToLargerAngle"	ToLargerAngle
"Max"	Max	"ToSmallerAngle"	ToSmallerAngle

Tabelle 5 „Beziehungen zwischen Bezeichnung und eindeutigem *Kommandoparameter*-Typ – Groß-/ Kleinschreibung ist zu beachten!“ (Quelle: selbst)

Die Methode `TSteering::ParsingCmd` ist (unter Anderem) für die Abbildung von *Bezeichnung* auf `TCmdId` zuständig. Eine Umkehrfunktion ist nicht erforderlich.

► **typedef struct { ... } TCmdTag**

repräsentiert ein einzelnes Kommando in einem Makro; besteht aus dem Typ des Kommandos und den Parametern; im folgenden **Kommandoinformation** genannt

Attribut in TCmdTag	Verwendung
► TCmdId Id PUBLIC	eindeutiger Typ des Kommandos, entsprechend diesem Typ wird ein Objekt erstellt, das die gewünschte Funktionalität durchführt
► TiParam P1 PUBLIC	erster <i>Kommandoparameter</i>
► TiParam P2 PUBLIC	zweiter <i>Kommandoparameter</i>
► char P3[_MAXLENP3+1] PUBLIC	dritter <i>Kommandoparameter</i> , der für die Übergabe von verschiedenen Werten (int, float, LPSTR und Kombinationen daraus) individuell pro Kommando interpretiert wird

Tabelle 6 „Attribute von TCmdTag“ (Quelle: selbst)



► **typedef enum { ... } TCCode**

gibt den Fortschritt der Kommandoverarbeitung an

TCCode	Verwendung
CFirstStep	weiter bei Methode FirstStep des Kommandos
CControlStep	weiter bei Methode ControlStep des Kommandos
CReadyStep	weiter bei Methode ReadyStep des Kommandos
CReady	weiter bei Methode Ready (gibt nur CReady zurück) des Kommandos kennzeichnet <u>Kommandoverarbeitung erfolgreich</u>
CMeasure	signalisiert Steering, dass die <u>Intensitätsmessung des aktuellen Detektors neu gestartet</u> werden soll
CRecall	<u>Kommandoausführung kurz unterbrechen</u> (Methode TSteering::StartTimer aufrufen)
CStop	wird nur an einer Stelle zurückgegeben, aber nie ausgewertet
CFailure	nicht verwendet; AUSKOMMENTIERT
CHalted	nicht verwendet; AUSKOMMENTIERT

Tabelle 7 „symbolische Werte von TCCode“ (Quelle: selbst)

III Konstanten

► **#define id_Report 10300**

GLOBAL

Ressourcen-ID eines Listenfeldes, in das TSteering::SendReport seine Informationen ausgeben kann, deklariert in RC_DEF.H

► **const UINT nMaxArg= 4**

GLOBAL

GEÄ

maximale Anzahl von berechenbaren Zwischenergebnissen – siehe TSteering::dCalcArg; kann nicht static deklariert werden

#define-Makros sollten vermieden werden, besser const-Deklaration KORRIGIERT

► **const UINT _MAXLENMAKFILENAME= 20**

GLOBAL

NEU

maximale Länge von TMacroTag::Filename

► **const UINT _MAXLENMACRONAME= 20**

GLOBAL

NEU

maximale Länge von TMacroTag::Name

► **const UINT _MAXLENP3= 50**

GLOBAL

NEU

maximale Länge von TCmdTag::P3



IV Klassen

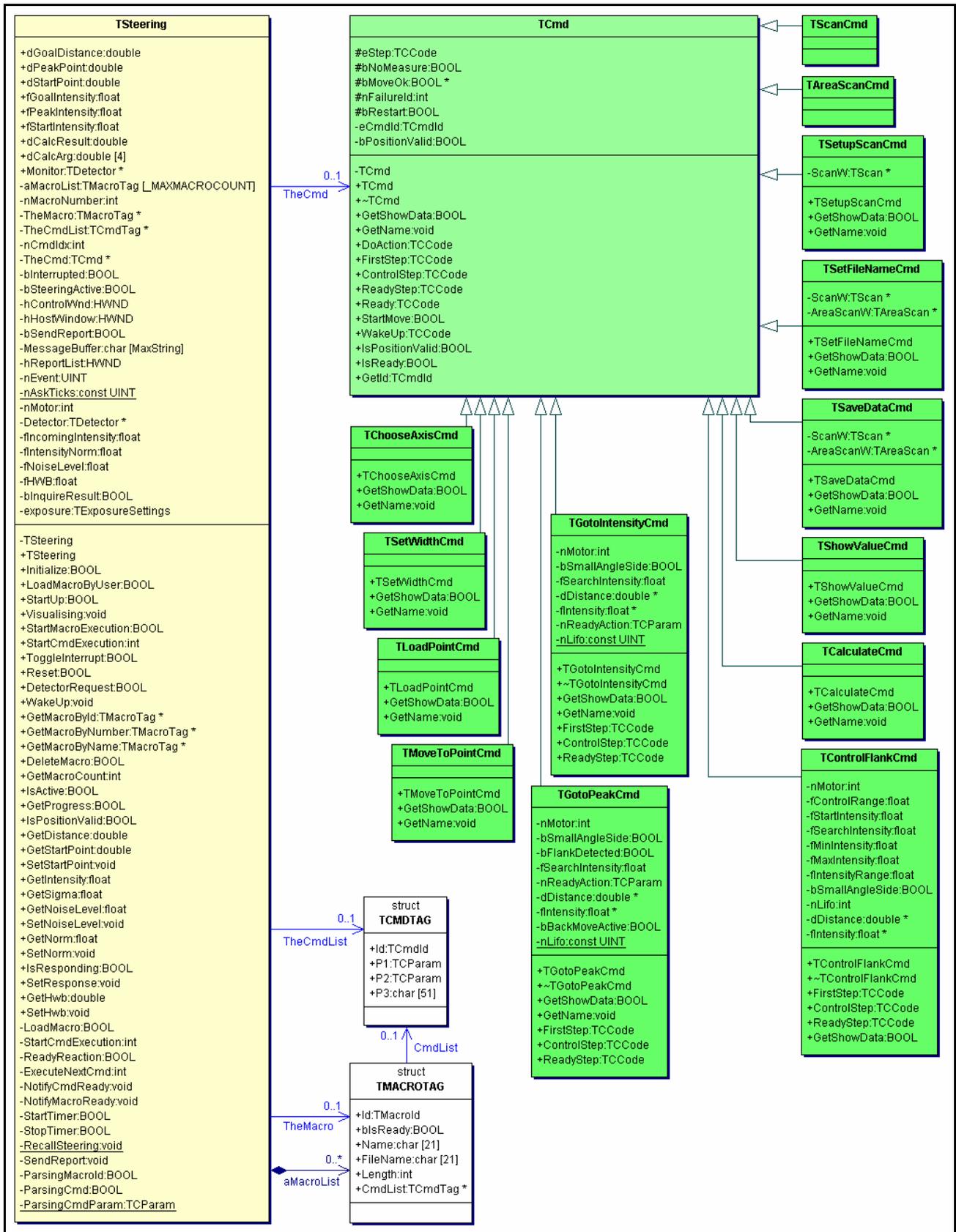


Abbildung 1 „UML-Klassendiagramm: Subsystem Ablaufsteuerung“ (Quelle: Together®, Version 6.0)



IV.1 Klasse TSteering

Deklaration : WORKFLOW.H

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TSteering könnten erzeugt werden. Zur Benutzung der Ablaufsteuerung kann nur das Steering-Objekt verwendet werden, d.h. es brauchen keine eigenen Objekte von TSteering erstellt werden! Tut man dies doch, kommt es sogar zu Fehlern, weil jedes TSteering-Objekt mit dem Hauptfenster kommuniziert. Dieses sendet seine Nachrichten jedoch stets an das o.g. Steering-Objekt.

IV.1.1 Friends

Methode/ Klasse	Gründe für die Friend-Relation
Methode RecallSteering	bTimerActive Friend-Relation nicht mehr erforderlich nachdem RecallSteering static deklariert wurde AUSKOMMENTIERT
Methode FrameWndProc	Methode WakeUp AUSKOMMENTIERT, nachdem Methode WakeUp public deklariert wurde
TCmd	nFailureId, bRestart
TMacroExecute	grundlos zum Friend erklärt AUSKOMMENTIERT

Tabelle 8 „Friends der Klasse TSteering“ (Quelle: selbst)

IV.1.2 Attribute

► **TMacroTag aMacroList[_MAXMACROCOUNT] PRIVATE GEÄ**

Liste der (mit Methode LoadMacro geladenen) Makros; mit den Informationen über das Makro (Typ, etc.) und den auszuführenden Kommandos; Initialisierung nicht notwendig, da nur die Anzahl der Elemente (nMacroNumber) initialisiert werden muss

Maximalgröße sollte über eine Konstante etwas variabler gestaltet werden. diese Konstante müsste dann auch bei Methode GetMacroByNumber, Zeile num > 19 beachtet werden; durch _MAXMACROCOUNT ersetzt

► **int nMacroNumber PRIVATE GEÄ**

Anzahl der Makros in aMacroList, das letzte Makro befindet sich bei Index aMacroNumber-1; im Konstruktor mit 0 initialisiert

► **TMacroTag* TheMacro PRIVATE**

Zeiger auf das derzeit ausgeführte Makro; im Konstruktor mit NULL initialisiert

► **TCmdTag* TheCmdList PRIVATE**

Liste der Kommandos von TheMacro; vor der Makroausführung (Methode StartMacroExecution) initialisiert

► **int nCmdIdx PRIVATE**

Index in TheCmdList des derzeit ausgeführten Kommandos; Wertebereich [0 ... TheMacro::Length-1]; im Konstruktor mit -1 initialisiert

**▶ TCmd* TheCmd****PRIVATE**

Entsprechend der Id des aktuellen Kommandos (`TheCmdList[nCmdIndx]->Id`) wird ein Objekt (abgeleitet von `TCmd`) erstellt, das die gewünschte Funktionalität des Kommandos durchführt. `TheCmd` ist ein Zeiger auf dieses Objekt. im Konstruktor mit `NULL` initialisiert.

▶ BOOL bInterrupted**PRIVATE GEÄ**

gibt an, dass die Makroverarbeitung gestoppt wurde; Stoppen und Fortsetzen der Makroausführung ist mit Methode `ToggleInterrupt` möglich. beim Start der Makroverarbeitung (Methode `StartCmdExecution`) mit `FALSE` initialisiert

▶ BOOL bReset**PRIVATE NEU**

gibt an, dass die Makroverarbeitung mit Methode `Reset` abgebrochen wurde; beim Start der Makroverarbeitung (Methode `StartCmdExecution`) mit `FALSE` initialisiert; bei Methode `Reset` auf `TRUE` gesetzt

▶ BOOL bSteeringActive**PRIVATE**

gibt an, ob derzeit ein Kommando ausgeführt wird; Bei `bInterrupted == TRUE`, ist `bSteeringActive == FALSE`, weil die Makroverarbeitung gestoppt wurde. im Konstruktor mit `FALSE` initialisiert

▶ HWND hControlWnd**PRIVATE**

Handle auf das Dialogfenster, das ein Listefeld für Informationsausgaben enthalten kann (siehe `hReportList`); Dieses Fenster empfängt die Botschaft `cm_CounterSet` vom (bei Methode `StartUp` gesetzten) Detektor. Daraufhin muss das Zählerfenster aktualisiert werden (Methode `TDetector::UpdateDisplay`) und `Steering` muss informiert werden, dass neue Messwerte vorliegen (`TSteering::DetectorRequest`) – **tut man dies nicht, hängt die Makroverarbeitung in einer Endlosschleife.**

▶ HWND hHostWindow**PRIVATE NEU**

Handle zu dem Fenster, das über die Windowsbotschaft `cm_SteeringReady` benachrichtigt werden soll, wenn die Makroverarbeitung beendet ist

▶ BOOL bSendReport**PRIVATE GEÄ**

bestimmt, ob bei Methode `SendReport` Informationen ausgegeben werden sollen (`TRUE`) oder nicht (`FALSE`); im Konstruktor mit `FALSE` initialisiert; bei Methode `Visualising` mit dem Wert des ersten Parameters überschrieben

▶ char MessageBuffer[MaxString]**PRIVATE**

Meldung die bei Methode `SendReport` – entweder in einem Listefeld (siehe `hReportList`) oder in der Statuszeile des Hauptfensters ‚Steuerprogramm‘ – ausgegeben werden soll

▶ HWND hReportList**PRIVATE GEÄ**

Handle zu einem Listefeld, in das der Inhalt von `MessageBuffer` (bei Methode `SendReport`) ausgegeben werden soll; kann `NULL` sein wenn die Informationen in der Statuszeile des Hauptfensters ‚Steuerprogramm‘ ausgegeben werden sollen; im Konstruktor mit `NULL` initialisiert und bei Methode `Visualising` ggf. mit dem Handle des Steuerelements mit Ressourcen-ID `::id_Report` überschrieben



- **UINT nEvent** **PRIVATE** **NEU**
Handle auf einen aktiven Timer in TSteering oder wenn TTSteering-Timer inaktiv ist 0. Darf nicht static sein, sonst verwendet Methode RecallSteering ein anderes nEvent als StartTimer und StopTimer – **Compiler-/ Linkerfehler?!**
- **static const UINT nAskTicks= 30** **PRIVATE** **NEU**
Diese Konstante gibt an, wie lange [in Millisekunden] die Kommandoverarbeitung dauern soll
- **int nMotor** **PRIVATE** **GEÄ**
Index eines ausgewählten Antriebs in Objekt lpMList; nur in den von TCmd abgeleiteten Klassen benutzt, wird jeweils vor der Benutzung initialisiert
- **double dGoalDistance** **PUBLIC**
entspricht der Absolutposition des ausgewählten Antriebs (in **Nutzereinheiten**), die bei Kommando TGotoIntensityCmd oder TGotoPeakCmd errechnet wurde; nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert im Konstruktor mit 0 initialisiert
- **double dPeakPoint** **PUBLIC**
Mit TLoadPointCmd kann die aktuelle Absolutposition des gewählten Antriebs (in **Nutzereinheiten**) – unter Anderem in dPeakPoint – zwischengespeichert werden. Mit anderen Kommandos können so gespeicherte Positionen auf dem Bildschirm ausgegeben oder wieder angefahren werden. nur in den von TCmd abgeleiteten Klassen benutzt
Dient derzeit nur zur Speicherung der Position des Intensitätsmaximums (siehe [5]: Abbildung 5).
wird nicht initialisiert⁵ im Konstruktor mit 0 initialisiert
- **double dStartPoint** **PUBLIC**
kann, wie dPeakPoint, dazu benutzt werden, die aktuelle Absolutposition zwischenzuspeichern; nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert im Konstruktor mit 0 initialisiert
- **TDevice* Device** **PRIVATE** **GEÄ**
Zeiger auf den ausgewählten Detektor; bei Methode StartUp initialisiert und irgendwo anders geschrieben
- **TDetector* Monitor** **PUBLIC** **NEU**
Verknüpfung auf einen Detektor für TScanCmd, TAreaScanCmd und TGotoPeakCmd
- **float fGoalIntensity** **PUBLIC**
entspricht der Intensität des ausgewählten Detektors am Zielpunkt der Bewegung; nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert, im Konstruktor mit 0 initialisiert

⁵ Es gibt Kommandos **(1)**, bei denen dieses Attribut nur gelesen wird. Bei anderen Kommandos **(2)** wird es auch geschrieben. Beim Aufruf eines **(1)**-Kommandos, vor einem **(2)**-Kommando, besteht die Gefahr einer nicht initialisierten Benutzung. Tipp: Zur Beginn der Makroausführung initialisieren



- **float fPeakIntensity** **PUBLIC**
dient zur Speicherung des Intensitätsmaximums (siehe [5]: Abbildung 5) des ausgewählten Detektors; nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert im Konstruktor mit 0 initialisiert
- **float fStartIntensity** **PUBLIC**
wird nur bei Methode TTopographyExecute::Dlg_OnCommand gelesen, aber nie geschrieben
- **float fIncomingIntensity** **PRIVATE**
enthält die aktuelle Intensität⁶ des ausgewählten Detektors
wird nicht initialisiert im Konstruktor mit 0 initialisiert
- **float fIntensityNorm** **PRIVATE**
ist ein Divisor, der zur Korrektur der aktuellen Intensität des ausgewählten Detektors (fIncomingIntensity) benutzt werden kann; wenn fIntensityNorm != 0, dann gilt $fIncomingIntensity := fIncomingIntensity / fIntensityNorm$ (sonst bleibt fIncomingIntensity unverändert)
im Konstruktor mit 0 initialisiert
- **float fNoiseLevel** **PRIVATE**
definiert das Rausch-Level; Intensitäten unterhalb von fNoiseLevel werden bei TGoToPeakCmd nicht beachtet. im Konstruktor mit 300 initialisiert
- **double dCalcResult** **PUBLIC**
Beim Aufruf von Kommando TCalculateCmd wird dCalcResult, je nachdem welchen der Werte (Difference, Opposite, Middle) der Parameter annimmt, berechnet (Absolutpositionen in Nutzereinheiten des ausgewählten Antriebs). nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert, im Konstruktor mit 0 initialisiert
- **double dCalcArg[nMaxArg]** **PUBLIC**
kann, wie dPeakPoint, dazu benutzt werden, die aktuelle Absolutposition zwischenspeichern; der zweite Parameter des TLoadPointCmd-Kommandos gibt an, an welchem Index die Position zu speichern ist; nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert im Konstruktor mit 0 initialisiert
- **float fHwB** **PRIVATE** **GEÄ**
beim Aufruf von Methode TCalculateCmd::TCalculateCmd(Hwb) wird die Halbwertsbreite (siehe [5]: Abbildung 5) berechnet (in Nutzereinheiten des ausgewählten Antriebs); nur in den von TCmd abgeleiteten Klassen benutzt
wird nicht initialisiert im Konstruktor mit 0 initialisiert
- **BOOL bInquireResult** **PRIVATE** **GEÄ**
nur ein Lesezugriff:
- Methode ExecuteNextCmd

⁶ Einheit ist abhängig vom Detektor



- **TExposureSettings exposure** **PRIVATE** **NEU**
speichert Detektorparameter vor Pausieren der Makroverarbeitung, um diese beim Fortsetzen wiederherstellen zu können
- **BOOL bTimerActive** **PRIVATE** **ENTF**
Flag, das kennzeichnet, ob der Timer aktiv ist (TRUE) oder nicht (FALSE); Starten bzw. Stoppen des Timer mit den API-Methoden `timeSetEvent` bzw. `timeKillEvent`. im Konstruktor mit FALSE initialisiert
Redundant und fehlerhaft implementiert: Mit `nEvent == 0` kann ein inaktiven Timer besser ermittelt werden. KORRIGIERT und AUSKOMMENTIERT
- **BOOL b_id_Angle** **PROTECTED** **ENTF**
toter Code AUSKOMMENTIERT
nur ein Schreibzugriff (was wird zugewiesen?):
 - Methode `Visualising`: Wert des zweiten Parameters
- **BOOL b_id_PeakIntensity** **PROTECTED** **ENTF**
toter Code AUSKOMMENTIERT
nur ein Schreibzugriff (was wird zugewiesen?):
 - Methode `Visualising`: Wert des dritter Parameters
- **BOOL b_id_Intensity** **PROTECTED** **ENTF**
toter Code AUSKOMMENTIERT
nur ein Schreibzugriff (was wird zugewiesen?):
 - Methode `Visualising`: Wert des vierten Parameters
- **BOOL b_id_HalfWidth** **PROTECTED** **ENTF**
toter Code AUSKOMMENTIERT
nur ein Schreibzugriff (was wird zugewiesen?):
 - Methode `Visualising`: Wert des fünften Parameters
- **BOOL b_id_Report** **PROTECTED** **ENTF**
toter Code AUSKOMMENTIERT
nur ein Lesezugriff
 - Methode `Visualising`: stattdessen kann der Wert des sechsten Parameters benutzt werden
nur ein Schreibzugriff (was wird zugewiesen?):
 - Methode `Visualising`: Wert des sechsten Parameters
- **int nFailureId** **PROTECTED** **ENTF**
toter Code; nur ein Schreibzugriff bei Methode `StartCmdExecution` (Verwechslung mit `TCmd::nFailureId`?) AUSKOMMENTIERT

IV.1.3 Methoden

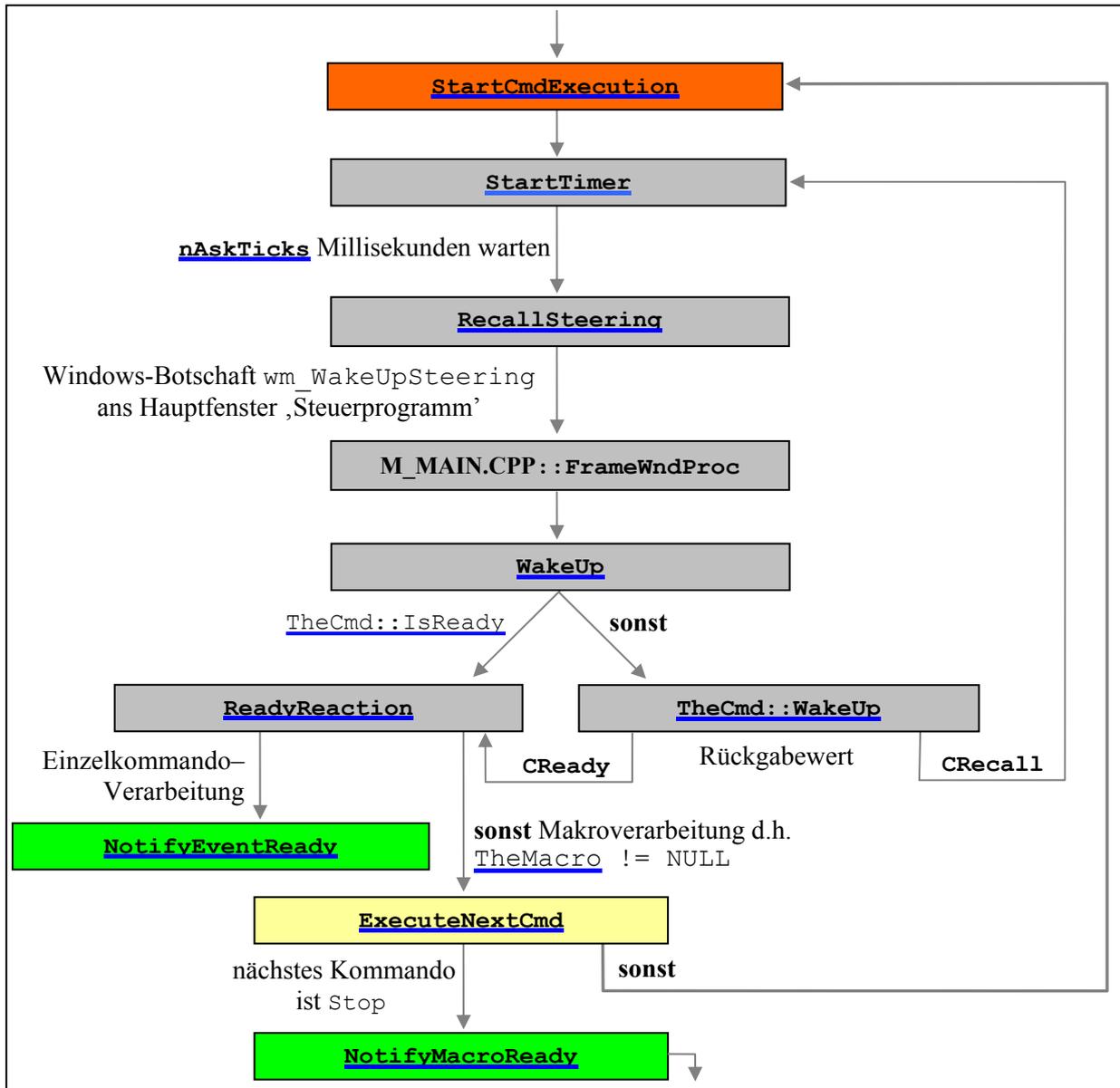


Abbildung 2 „Methodenaufufe während der Einzelkommando- bzw. Makroverarbeitung“ (Quelle: selbst)

Die grau hinterlegten Methoden ermöglichen eine Pausierung der Kommandoverarbeitung. Die blau unterstrichenen Member sind Teil des Steering-Objekts.

► TSteering(void)

PUBLIC

Der Konstruktor initialisiert einen kleinen Teil der Attribute. Diese sind unter IV.1.2 Attribute besonders gekennzeichnet.

Es wäre sehr wichtig sicherzustellen, dass maximal ein Objekt vom Typ TSteering erzeugt werden kann, da der Timer-Rückrufmechanismus (RecallSteering) und TCmd (und abgeleitete Klassen) nur auf dem Steering-Objekt arbeiten. Copykonstruktor bereits leer implementiert.

► TSteering(TSteering&)

PRIVATE

NEU

leer implementiert, um das Kopieren von TSteering-Objekten zu verhindern



► **BOOL Initialize(const HWND, PUBLIC
const int, const int)**

deaktiviert (durch den Aufruf von `Visualising(0, 0, 0, 0, 0, 0)`) Meldungsausgaben bei Methode `SendReport` und initialisiert die Makroverarbeitung, indem die drei Parameter an Methode `StartUp` weitergegeben werden. Anschließend werden die in [Tabelle 2](#) genannten Makros aus `STANDARD.MAK` geladen (Methode `LoadMacro`). Sobald das Laden eines Makros fehlschlägt, wird `FALSE` zurückgegeben, sonst `TRUE`.

Die aufgerufene Methode `Visualising` verwendet `hControlWnd`, obwohl dieses erst bei Methode `StartUp` initialisiert wird.

► **BOOL LoadMacroByUser(void) PUBLIC**

lädt die in [Tabelle 2](#) genannten Makros aus `SCAN.MAK`; Sobald das Laden eines Makros fehlschlägt, wird `FALSE` zurückgegeben, sonst `TRUE`.

► **BOOL StartUp(const HWND, PUBLIC
const int, TDetector*)**

der erste Parameter wird `hControlWnd` zugewiesen; der zweite Parameter entspricht dem Index des Antriebs, der ausgewählt werden soll (siehe [\[1\]](#): `mlSetAxis`); der dritte Parameter entspricht dem Index des auszuwählenden Detektors (siehe [\[3\]](#): `TDList::DP(int)`)

Rückgabewert stets `TRUE`

► **void Visualising(BOOL, BOOL) PUBLIC GEÄ**

Die Parameter regeln, wie die Meldungen bei Methode `SendReport` ausgegeben werden. Der erste Parameter gibt an, ob Meldungen ausgegeben werden sollen (wird `bSendReport` zugewiesen). Der zweite Parameter regelt, wo die Meldungen ausgegeben werden (siehe Methode `SendReport`):

- `TRUE` → `hReportList` wird mit dem Handle des Steuerelements mit Ressourcen-Id `id_Report` initialisiert, das sich im Fenster mit dem Handle `hControlWnd` befinden muss
- `FALSE` → wird `hReportList` mit `NULL` initialisiert

Parameter zwei bis fünf waren totter Code; Wertebereich `BOOL` für die anderen beiden Parameter ausreichend KORRIGIERT

► **BOOL StartMacroExecution(TMacroTag*, HWND) PUBLIC GEÄ**

prüft ob der erste Parameter verschieden von `NULL` ist, sonst wird `FALSE` zurückgegeben; Wenn bereits ein Makro ausgeführt wird, ist der Rückgabewert ebenfalls `FALSE`. Sonst wird der Detektor eingerichtet, das (als ersten Parameter) übergebene Makro wird ausgeführt, der zweite Parameter wird `hHostWindow` zugewiesen und `TRUE` zurückgegeben. siehe [Abbildung 2](#)

Wertebereich `BOOL` des Rückgabewerts ausreichend KORRIGIERT

► **int StartCmdExecution(TCmdId, int, int, PUBLIC GEÄ
LPSTR, HWND)**

erzeugt ein neues Kommando: der erste Parameter ist der Typ (`TCmd::Id`), der zweite bis vierte Parameter sind der erste bis dritte *Kommandoparameter* (`TCmd::P1` bis `TCmd::P3`); der letzte Parameter wird `hControlWnd` zugewiesen; anschließend wird dieses Kommando ausgeführt und der Rückgabewert von `StartCmdExecution(TCmdTag)` zurückgegeben

Wertebereich vom zweiten und dritten Parameters sollte `TCParam` sein. Kann nicht korrigiert werden, weil an einigen Stellen `Omega2ThetaScan` (`typedef enum TxScanType`) angegeben wird (`TxScanType` und `TCParam` sind disjunkt)!



► **int StartCmdExecution(TCmdTag)** **PRIVATE** **GEÄ**

Wenn das (als Parameter übergebene) Kommando NULL ist, wird FALSE zurückgegeben. sonst wird der Detektor eingerichtet und ein dem Kommandotyp entsprechendes Objekt erzeugt, das die gewünschte Funktionalität durchführt. Bei einem unbekanntem Kommandotyp oder bei Fehlern während der Kommandoausführung wird FALSE zurückgegeben, sonst wird Methode StartTimer aufgerufen (um die Kommandoausführung für einige Augenblicke zu pausieren) und TRUE zurückgegeben. siehe [Abbildung 2](#)

► **int ExecuteNextCmd(void)** **PRIVATE**

Sobald das erste Kommando bei Methode StartMacroExecution erfolgreich durchgeführt und eine kleine Pause gemacht wurde (siehe [Abbildung 2](#)), wird das nächste Kommando ausgeführt (auch Sprünge werden unterstützt). Für das nächste Kommando wird wiederum Methode ExecuteNextCmd verwendet. Nachdem das nächste Kommando gestartet wurde, wird TRUE zurückgegeben; ist das Kommando vom Typ Stop, wird FALSE zurückgegeben.

► **BOOL StartTimer(BOOL)** **PRIVATE** **GEÄ**

versucht einen Timer zu starten; Bei Erfolg enthält nEvent das Handle zum Timer, es wird TRUE zurückgegeben. Schlägt dies fehl, ist nEvent==0 und es wird FALSE zurückgegeben. Wenn der Timer aktiv ist, wird einmalig die Methode RecallSteering aufgerufen, danach stoppt der Timer – Parameter TIME_ONESHOT. siehe [Abbildung 2](#)

Der Parameter (Wertebereich BOOL ausreichend) gibt an, ob nAskIdx dekrementiert (FALSE) oder auf nMaxAskNumber gesetzt (TRUE) wird. **AUSKOMMENTIERT, da keine Lesezugriff auf nAskIdx bTimerActive erhält den Wert TRUE, bevor klar ist, ob ein Timer installiert werden kann. nEvent sollte stattdessen dazu verwendet werden, einen aktiven Timer zu identifizieren. KORRIGIERT**

► **BOOL StopTimer(void)** **PRIVATE**

stoppt den Timer, falls dieser aktiv ist; setzt nEvent auf 0; Rückgabewert stets TRUE

► **static void CALLBACK RecallSteering (UINT, UINT, DWORD, DWORD, DWORD)** **PRIVATE** **NEU**

wird beim Aufruf des Timers gerufen; setzt nEvent = 0 und kennzeichnet damit, dass der Timer nun inaktiv ist; Anschließend wird die Windows-Botschaft wm_WakeUpSteering ans Hauptfenster ‚Steuerprogramm‘ geschickt. Dort wird Methode Steering->WakeUp aufgerufen. siehe [Abbildung 2](#)

► **void WakeUp(void)** **PUBLIC** **GEÄ**

siehe [Abbildung 2](#); zusätzlich wird auch der Rückgabewert CMeasure von Methode TheCmd::WakeUp ausgewertet, dann wird die Detektormessung neu gestartet und Statusinformationen des bearbeiteten Kommandos werden ausgegeben (Methode SendReport)

► **BOOL DetectorRequest(void)** **PUBLIC**

ermittelt die aktuelle Intensität des ausgewählten Detektors und dividiert sie durch fIntensityNorm; das Ergebnis wird in fIncomingIntensity gespeichert. Anschließend wird die Funktionalität von Methode WakeUp durchgeführt. siehe auch hControlWnd
Rückgabewert ist stets TRUE

► **BOOL ReadyReaction(void)** **PRIVATE**

siehe [Abbildung 2](#); zusätzlich werden Statusinformationen des bearbeiteten Kommandos ausgegeben (Methode SendReport)



► **void NotifyMacroReady(void)** **PRIVATE** **GEÄ**
stoppt den Detektor und benachrichtigt das Fenster mit dem Handle `hHostWindow` (über die Windowsbotschaft `cm_SteeringReady`), dass die Ausführung des Makros beendet ist; siehe [Abbildung 2](#)

► **void NotifyCmdReady(void)** **PRIVATE** **GEÄ**
Quellcode ist völlig identisch zu Methode `NotifyMacroReady`; wird jedoch aufgerufen, um die Beendigung eines Einzelkommandos zu verkünden; siehe [Abbildung 2](#)

► **BOOL ToggleInterrupt(void)** **PUBLIC**
Bei `bInterrupted == FALSE` wird die Makroausführung gestoppt. Anschließend (`bInterrupted == TRUE`) kann die Ausführung ab diesem Punkt fortgesetzt werden, wenn Methode `ToggleInterrupt` erneut aufgerufen wird. gibt den neuen Wert von `bToggleInterrupt` zurück, d.h. Aktion erfolgreich?

► **BOOL Reset(void)** **PUBLIC**
bricht die Verarbeitung des aktuellen Makros ab; Makro kann später nicht fortgesetzt werden; Rückgabewert stets `TRUE`

Absturz, wenn kurz nach Beginn der Makroverarbeitung gestoppt wird KORRIGIERT

► **TMacroTag* GetMacroById(TMacroId)** **PUBLIC**
sucht in `aMacroList` nach einem Makro mit dem (als Parameter übergeben) Typ; bei Erfolg wird ein Zeiger auf das gefundene Makro zurückgegeben, sonst `NULL`

► **TMacroTag* GetMacroByNumber(int)** **PUBLIC**
gibt einen Zeiger auf das Makro aus `aMacroList` zurück (den Index bestimmt der erste Parameter); Rückgabewert ist `NULL`, wenn der Parameter den Wertebereich `[0 ... nMacroNumber-1]` verletzt

bei der Prüfung des Wertebereichs können Fehler auftreten KORRIGIERT

► **TMacroTag* GetMacroByName(LPSTR)** **PUBLIC**
wie Methode `GetMacroById`, hier wird jedoch ein Makro mit dem (als Parameter übergebenen) Namen gesucht

► **BOOL DeleteMacro(TMacroTag*)** **PUBLIC**
löscht das erste Makro aus `aMacroList`, das den Namen des (als Parameter übergebenen) Makros besitzt und verringert `nMacroNumber` um 1; `FALSE` wird zurückgegeben, wenn der Parameter `NULL` ist, sonst `TRUE`

Wenn das Makro nicht vorhanden ist, kommt es zum Absturz. KORRIGIERT

► **int GetMacroCount(void)** **PUBLIC**
gibt die Anzahl der Makros in `aMacroList` zurück (entspricht `nMacroNumber`)

► **BOOL IsActive(void)** **PUBLIC** **NEU**
gibt zurück, ob die Makroverarbeitung aktiv ist (nicht pausiert, nicht abgebrochen, nicht fertig)

► **BOOL IsReset(void)** **PUBLIC** **NEU**
gibt zurück, ob die Makroverarbeitung mit Methode `Reset` abgebrochen wurde



- ▶ **BOOL GetProgress (int&, LPSTR, int&)** **PUBLIC NEU**
wenn die Makroverarbeitung nicht aktiv ist (`IsActive() == FALSE`), wird `FALSE` zurückgegeben; sonst wird im ersten Parameter der Index des aktuellen Schritts (`nCmdIdx`), im zweiten der Fortschritt des aktuellen Kommandos, im dritten die Anzahl der Schritt während der Makroverarbeitung zurückgeben, Rückgabewert ist `TRUE`

- ▶ **BOOL IsPositionValid(void)** **PUBLIC**
gibt an, ob die Positionierung des ausgewählten Antriebs erfolgreich war (entspricht `bPositionValid` des aktuellen Kommandos)

- ▶ **double GetDistance(void)** **PUBLIC**
gibt die aktuelle Absolutposition (in *Nutzereinheiten*) des ausgewählten Antriebs zurück

- ▶ **double GetStartPoint (void)** **PUBLIC NEU**
gibt die `dStartPoint` zurück

- ▶ **void SetStartPoint (double)** **PUBLIC NEU**
setzt `dStartPoint` auf den (als Parameter übergebenen) Wert

- ▶ **float GetIntensity(void)** **PUBLIC**
gibt die aktuelle Intensität⁷ des ausgewählten Detektors zurück (entspricht `fIncomingIntensity`)

- ▶ **float GetSigma(void)** **PUBLIC**
gibt den Faktor `fSigma` des ausgewählten Detektors zurück (entspricht einem Wert, der aus der aktuellen Intensität berechnet wird, siehe Methode `TDevice::CalculateSigma`); Quelle: [3]

- ▶ **float GetNoiseLevel(void)** **PUBLIC**
gibt das Rausch-Level (`fNoiseLevel`) zurück

- ▶ **void SetNoiseLevel(float)** **PUBLIC**
setzt das Rausch-Level (`fNoiseLevel`) auf den (als Parameter übergebenen) Wert

- ▶ **float GetNorm(void)** **PUBLIC**
gibt den Divisor, der zur Korrektur der aktuellen Intensität verwendet werden kann zurück (entspricht `fIntensityNorm`)

- ▶ **void SetNorm(float)** **PUBLIC**
setzt den Divisor, der zur Korrektur der aktuellen Intensität verwendet werden kann zurück auf den (als Parameter übergebenen) Wert (entspricht `fIntensityNorm`)

- ▶ **BOOL IsResponding(void)** **PUBLIC**
gibt an, ob Methode `SendReport` Meldungen ausgeben soll (entspricht `bSendReport`)

⁷ Einheit ist abhängig vom Detektor



- **void SetResponse(BOOL)** **PUBLIC**
weist den ersten Parameter `bSendReport` zu (gibt an, ob Methode `SendReport` Meldungen ausgeben soll)
- **double GetHwb (void)** **PUBLIC** **NEU**
gibt `fHwb` (bei `TCalculateCmd` berechnete Halbwertsbreite) zurück
- **void SetHwb (double)** **PUBLIC** **NEU**
setzt `fHwb` auf den (als Parameter übergebenen) Wert
- **BOOL LoadMacro(LPSTR, LPSTR)** **PRIVATE** **GEÄ**
lädt das Makro mit dem Namen des ersten Parameters aus der `mak`-Datei mit dem Namen des zweiten Parameters (der Dateiname darf keinen Pfad beinhalten und muss auf „MAK“ enden; die Datei wird in dem Verzeichnis gesucht, wo die Anwendung gestartet wird). Bei Erfolg wird das Makro in `aMacroList` eingefügt – wenn bereits ein Makro dieses Typs (`TMacroId`) existiert, wird dieses gelöscht – anschließend wird `nMacroNumber` um eins erhöht.
Die Methode gibt `TRUE` zurück, wenn das Makro erfolgreich geladen wurde – `FALSE` in jeden erdenklichen Fehlerfall (Datei oder Makro nicht gefunden oder Syntaxfehler).
Nicht in jedem Fall wird das Dateihandle wieder freigegeben. Zum Löschen eines vorhandenen Makros, sollte Methode `DeleteMacro` benutzt werden. KORRIGIERT
- **void SendReport(void)** **PRIVATE** **GEÄ**
wenn `bSendReport == FALSE` oder `MessageBuffer == NULL` ist, wird keine Meldung ausgegeben; wenn `hReportList != NULL`, wird Meldung `MessageBuffer` am Ende des Listenfelds hinzugefügt, sonst wird die Meldung im Hauptfenster ‚Steuerprogramm‘ ausgegeben
- **BOOL ParsingMacroId(TMacroTag&, LPSTR)** **PRIVATE**
übersetzt die, als Parameter übergebene, „beliebige“, textuelle Bezeichnung eines Makros in dessen Typ und speichert diese bei Erfolg im Makro (das als erster Parameter übergeben wird); Wenn ein Makro diesen Typs bereits in `aMacroList` vorhanden ist oder die Bezeichnung nicht übersetzt werden kann, wird `FALSE` zurückgegeben, sonst `TRUE`. siehe [Tabelle 2](#)
Schlechter Programmierstil, durch die nicht benötigte, methodenumschließende while (1) – Schleife wird eine Endlosschleife suggeriert. KORRIGIERT
- **BOOL ParsingCmd(TCmdTag&, LPSTR, LPSTR, LPSTR, LPSTR)** **PRIVATE**
übersetzt die (als ersten Parameter übergebene) „beliebige“, textuelle Bezeichnung eines Kommandos in dessen Typ; die Parameter drei bis fünf werden (abhängig von diesem Typ) als *Kommandoparameter* interpretiert, die bei Erfolg (zusammen mit dem Typ) im ersten Parameter abgespeichert werden. Wenn die Bezeichnung nicht übersetzt werden konnte oder einer der Parameter fehlt oder ungültig ist, wird eine Meldung ausgegeben und `FALSE` zurückgegeben, sonst `TRUE`. siehe [Tabelle 4](#)
Schlechter Programmierstil, durch die nicht benötigte, methodenumschließende while (1) – Schleife wird eine Endlosschleife suggeriert. KORRIGIERT
- **TCPParam ParsingCmdParam(LPSTR param)** **PRIVATE**
übersetzt die (als ersten Parameter übergebene) „beliebige“, textuelle Bezeichnung von ausgewählten Parametern in dessen Typ und gibt ihn zurück (bei fehlerhaftem oder unbekanntem Parameter wird 0 zurückgegeben); siehe [Tabelle 5](#)



- **BOOL GetFileLine(FILE*, LPSTR, int)** **PRIVATE** **ENTF**
liest eine Zeile aus der Datei mit dem Handle des ersten Parameters in die (als zweiten Parameter übergebene) Zeichenkette; Die Maximallänge des dritten Parameters wird dabei nicht überschritten.
Rückgabewert TRUE ↔ Zeile fehlerfrei gelesen, Maximallänge wurde nicht überschritten
nachdem **Probleme beim Lesen von Textdateien mit UNIX-Zeilenumbrüchen** festgestellt wurden;
AUSKOMMENTIERT und durch Klasse zum Lesen von Textdateien ersetzt
- **HWND GetControlWnd(void)** **PUBLIC** **ENTF**
gibt das Fensterhandle hControlWnd zurück
nur einmal private verwendet, kann entfernt werden; AUSKOMMENTIERT
- **int TimerRequest(void)** **PRIVATE** **ENTF**
deklariert aber nicht implementiert; AUSKOMMENTIERT
- **BOOL IsMacroCalled(void)** **PRIVATE** **ENTF**
toter Code (Quelle: [6]); **AUSKOMMENTIERT**
- **int ReturnMacroCall(void)** **PRIVATE** **ENTF**
toter Code (Quelle: [6]); **AUSKOMMENTIERT**
- **int CallMacro(TMacroTag*)** **PUBLIC** **ENTF**
toter Code (Quelle: [6]); **AUSKOMMENTIERT**



IV.1.4 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	1.241
„LOC of Implementation“ ⁸	LOCI	998
„LOC of Declaration“	LOCD	243
„Number Of Attributes“	NOA (0, 30)	32
„Number Of Operations“	NOO (0, 50)	44
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	76
„Number Of Constructors“	NOCON (0, 5)	2
„Number Of Overridden Methods“	NOOM (0, 10)	0
„Percentage of Private Members“	PPrivM	47
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	53
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	169
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	5
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	293
„True Comment Ratio“	TCR (5, 400)	29

Tabelle 9 „ausgewählte Metriken der Klasse TSteering“ (Quelle: Together[®], Version 6.0)

⁸ Diese Metriken sind nicht Bestandteil von Together, sondern wurden manuell ermittelt.



IV.2 Klasse TCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Obwohl es keine abstrakten Methoden gibt, werden nur Objekte von den von TCmd abgeleiteten Klassen (Spezialisierungen) erstellt.

IV.2.1 Friends

Klasse	Gründe für die Friend-Relation
TSteering	nFailureId, bRestart

Tabelle 10 „Friends der Klasse TCmd“ (Quelle: selbst)

IV.2.2 Attribute

► **TCCode eStep**

PROTECTED

gibt an, welcher Schritt als nächstes (bei der Kommandoverarbeitung) ausgeführt werden soll; siehe [Tabelle 7](#) **wird nicht initialisiert; im Konstruktor mit CReady initialisiert**

► **BOOL bNoMeasure**

PROTECTED

gibt an, dass keine Intensitätsmessung des ausgewählten Detektors stattfinden soll, wenn Methode WakeUp das nächste Mal aufgerufen wird; im Konstruktor mit FALSE initialisiert aber im Konstruktor fast aller abgeleiteten Klassen mit TRUE initialisiert

► **BOOL* bMoveOk**

PROTECTED

Liste von BOOL-Werten, die Anzahl der Elemente entspricht der Anzahl der angeschlossenen Antriebe (siehe [\[1\]](#): mlGetAxisNumber); Jedes Element gibt an, ob sich der Antrieb (der diesem Index entspricht) bewegt. Liste wird im Konstruktor dynamisch erzeugt und im Destruktor freigegeben

► **int nFailureId**

PROTECTED

kennzeichnet verschiedene Fehlerzustände während der Kommandoverarbeitung – die Bedeutung variiert von Kommando zu Kommando (Die leeren Zellen geben an, dass der angegebene Wert bei dieser Klasse nicht verwendet wird.):

Wert von nFailure	0	1	11
TSaveDataCmd	es ist kein Fehler aufgetreten	speichern der Datendatei fehlgeschlagen	
TChooseAxisCmd		der als Parameter angegebene Antrieb konnte nicht ausgewählt werden	
TChooseDeviceCmd		der als Parameter angegebene Detektor konnte nicht ausgewählt werden	
TGotoIntensityCmd			die zu suchende Intensität (fSearchIntensity) ist (wider erwartend) 0
sonst			

Tabelle 11 „Bedeutung der Werte in den Klassen, wo nFailure benutzt wird“ (Quelle: selbst)

die Makroverarbeitung in Methode TSteering::StartCmdExecution wird abgebrochen, wenn nFailure auf einen von 0 verschiedenen Wert setzt; im Konstruktor mit 0 initialisiert

**▶ BOOL bRestart****PROTECTED**

gibt an, dass die Makroverarbeitung mit Methode `TSteering::ToggleInterrupt` unterbrochen und soeben wieder fortgesetzt wurde; wird nur bei `TScanCmd` und `TAreaScanCmd` verwendet, dort im Konstruktor jeweils mit `FALSE` initialisiert

▶ TCmdId eCmdId**PRIVATE**

speichert den Kommando-Typ; im Konstruktor mit dem Typ der (als Parameter übergebenen) *Kommandoinformationen* initialisiert

▶ BOOL bPositionValid**PRIVATE**

gibt an, ob die Positionierung des ausgewählten Antriebs erfolgreich war; im Konstruktor mit `TRUE` initialisiert

IV.2.3 Methoden

▶ TCmd(TCmdTag)**PUBLIC**

im Konstruktor wird `bMoveOk` dynamisch erzeugt und ein Großteil der unter [IV.2.2 Attribute](#) initialisiert **nun werden alle initialisiert**

▶ TCmd(TCmd&)**PRIVATE****NEU**

leer implementiert, um das Kopieren von `TCmd` (und abgeleiteten) Objekten zu verhindern

▶ virtual ~TCmd(void)**PUBLIC**

gibt die dynamisch erzeugte Liste `bMoveOk` wieder frei

▶ TCCode DoAction(void)**PUBLIC**

setzt die Verarbeitung des Kommandos im nächsten Schritt fort – je nach `eStep` (`CFirstStep`, `CControlStep`, `CReadyStep` bzw. `CReady`); Der Rückgabewert einer aufgerufenen Methode (`FirstStep`, `ControlStep`, `ReadyStep` bzw. `Ready`) wird zurückgegeben.

▶ virtual TCCode FirstStep(void)**PUBLIC****▶ virtual TCCode ControlStep(void)****PUBLIC****▶ virtual TCCode ReadyStep(void)****PUBLIC****▶ virtual TCCode Ready(void)****PUBLIC**

geben hier alle nur `CReady` (Kommandoverarbeitung erfolgreich beendet) zurück

Die abgeleiteten Klassen können hier den nächsten Schritt der Kommandoverarbeitung implementieren und zurückgeben welches der nächste, zu verarbeitenden Schritt ist.

▶ virtual BOOL GetShowData(LPSTR)**PUBLIC**

gibt hier nur `TRUE` zurück

In den abgeleiteten Klassen überschrieben, dort wird – je nach Fehlerzustand der in `nFailureId` abgelegt ist – eine Fehlermeldung im ersten Parameter platziert und stets `TRUE` zurückgegeben. Wird nur von `TSteering` aufgerufen, um den Grund für die fehlgeschlagene Kommandoausführung oder Statusinformationen ausgeben zu können.

In allen abgeleiteten Klassen wurden Statusinformationen in Deutsch bzw. Englisch (abhängig von `GermanVersion`) eingefügt oder ergänzt.



► **virtual void GetName(LPSTR aName) Kull-** **PUBLIC** **NEU**
gibt im ersten Parameter den Namen der Klasse zurück, hier nur „BASE“

In den abgeleiteten Klassen überschrieben, um dort den jeweiligen Namen zurückzugeben; ignoriert wird das führende „T“ und das anschließende „Cmd“; bei TChooseAxisCmd wird demnach „ChooseAxis“ zurückgegeben.

► **BOOL StartMove(const int, double)** **PUBLIC**
bewegt den Antrieb mit dem Index der als erster Parameter angegeben wird zur Absolutposition (in **Nutzereinheiten**) im zweiten Parameter; nach erfolgreicher Positionierung wird TRUE – sonst FALSE – zurückgegeben

► **virtual TCCode WakeUp(void)** **PUBLIC**
wird aufgerufen, nachdem die Kommandoausführung pausiert wurde; prüft, ob die durch das Kommando gestarteten Antriebe mittlerweile still stehen – ist dies nicht der Fall, wird CRecall zurückgegeben; Wenn die Antriebe stehen wird entweder der Rückgabewert von Methode DoAction (wenn bNoMeasure == TRUE) oder CMeasure zurückgegeben. siehe [Abbildung 2](#)

► **BOOL IsPositionValid(void)** **PUBLIC**
gibt bPositionValid zurück; gibt an, ob die Positionierung des ausgewählten Antriebs erfolgreich war

► **BOOL IsReady(void)** **PUBLIC**
gibt zurück, ob eStep == CReady ist (Kommandoverarbeitung erfolgreich beendet)

► **TCmdId GetId(void)** **PUBLIC**
gibt eStep (was ist der nächste Schritt der der Kommandoverarbeitung) zurück

► **virtual BOOL DeviceRequest(void)** **PUBLIC** **ENTF**
toter Code; nur Rückgabe von TRUE (Quelle: [6]); **AUSKOMMENTIERT**

► **virtual LPSTR GetFailure(void)** **PUBLIC** **ENTF**
toter Code; nur Rückgabe von "Fehler" (Quelle: [6]); **AUSKOMMENTIERT**



IV.2.4 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	157
„LOC of Implementation“	LOCI	96
„LOC of Declaration“	LOCD	61
„Number Of Attributes“	NOA (0, 30)	7
„Number Of Operations“	NOO (0, 50)	12
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	19
„Number Of Constructors“	NOCON (0, 5)	2
„Number Of Overridden Methods“	NOOM (0, 10)	0
„Percentage of Private Members“	PPrivM	14
„Percentage of Protected Members“	PProtM (0, 10)	23
„Percentage of Public Members“	PPubM	64
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	55
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	2
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	46
„True Comment Ratio“	TCR (5, 400)	65

Tabelle 12 „ausgewählte Metriken der Klasse TCmd“ (Quelle: Together®, Version 6.0)



IV.3 Klasse TChooseAxisCmd

Deklaration : M_STEERH.CPP

Implementation : M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TChooseAxisCmd können erzeugt werden.

IV.3.1 Methoden

► **TChooseAxisCmd(TCmdTag)** **PUBLIC**

wählt die Antriebsachse (die als erster *Kommandoparameter* in den *-informationen* übergebenen wird) aus (gültige Werte: TAxisType); Bei Misserfolg wird nFailureId 1 gesetzt. Anschließend ist das Kommando beendet, eStep wird CReady gesetzt.

► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**

überschrieben um Fehlerinformationen (nFailureId != 0) oder Statusinformationen zurückzugeben; Rückgabewert stets TRUE

IV.3.2 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
,Lines Of Code' inkl. Leer- und Kommentarzeilen		LOC (0, 1000)	39
,LOC of Implementation'		LOCI	29
,LOC of Declaration'		LOCD	10
,Number Of Attributes'		NOA (0, 30)	0
,Number Of Operations'		NOO (0, 50)	2
,Number Of Members' Attribute + Methoden		NOM = NOA + NOO	2
,Number Of Constructors'		NOCON (0, 5)	1
,Number Of Overridden Methods'		NOOM (0, 10)	2
,Percentage of Private Members'		PPrivM	0
,Percentage of Protected Members'		PProtM (0, 10)	0
,Percentage of Public Members'		PPubM	100
,Weighted Methods Per Class'		WMPC1 (0, 30)	
Attribute			
,Attribute Complexity'		AC	0
Methoden			
,Maximum Number Of Parameters'		MNOP (0, 4)	1
,Cyclomatic Complexity'		CC	
Kommentare			
,Number Of Comments'		NOC	7
,True Comment Ratio'		TCR (5, 400)	18

Tabelle 13 „ausgewählte Metriken der Klasse TChooseAxisCmd“ (Quelle: Together[®], Version 6.0)

IV.4 Klasse TSetWidthCmd

Deklaration : M_STEERH.CPP

Implementation : M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TSetWidthCmd können erzeugt werden.



IV.4.1 Methoden

► TSetWidthCmd(TCmdTag)**PUBLIC**

setzt die Schrittweite (die als dritter *Kommandoparameter* in den *-informationen* übergebenen wird) für den ausgewählten Antrieb (gültige Werte: float); Anschließend ist das Kommando beendet, eStep wird CReady gesetzt.

► virtual BOOL GetShowData(LPSTR)**PUBLIC**

überschrieben um Statusinformationen zurückzugeben; Rückgabewert stets TRUE

IV.4.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
‚Lines Of Code‘ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	31
‚LOC of Implementation‘	LOCI	21
‚LOC of Declaration‘	LOCD	10
‚Number Of Attributes‘	NOA (0, 30)	0
‚Number Of Operations‘	NOO (0, 50)	2
‚Number Of Members‘ Attribute + Methoden	NOM = NOA + NOO	2
‚Number Of Constructors‘	NOCON (0, 5)	1
‚Number Of Overridden Methods‘	NOOM (0, 10)	2
‚Percentage of Private Members‘	PPrivM	0
‚Percentage of Protected Members‘	PProtM (0, 10)	0
‚Percentage of Public Members‘	PPubM	100
‚Weighted Methods Per Class‘	WMPC1 (0, 30)	
Attribute		
‚Attribute Complexity‘	AC	0
Methoden		
‚Maximum Number Of Parameters‘	MNOP (0, 4)	1
‚Cyclomatic Complexity‘	CC	
Kommentare		
‚Number Of Comments‘	NOC	7
‚True Comment Ratio‘	TCR (5, 400)	23

Tabelle 14 „ausgewählte Metriken der Klasse TSetWidthCmd“ (Quelle: Together[®], Version 6.0)

IV.5 Klasse TLoadPointCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TLoadPointCmd können erzeugt werden.



IV.5.1 Methoden

► **TLoadPointCmd(TCmdTag)** **PUBLIC**

Abhängig vom ersten *Kommandoparameter* in den, als Parameter übergebenen, *-informationen* wird TSteering::dGoalDistance in einem anderen Attribut von TSteering gespeichert.

erster Kommandoparameter	TSteering::dGoalDistance gespeichert in
Argument	TSteering::dCalcArg [<zweiter Kommandoparameter>]
Start	TSteering::dStartPoint
Peak	TSteering::dPeakPoint

Tabelle 15 „Möglichkeiten bei der Arbeitsweise von TLoadPointCmd“ (Quelle: selbst)

Anschließend ist das Kommando beendet, eStep wird CReady gesetzt.

► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**

überschrieben um Statusinformationen zurückzugeben; Rückgabewert stets TRUE

IV.5.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
,Lines Of Code' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	43
,LOC of Implementation'	LOCI	33
,LOC of Declaration'	LOCD	10
,Number Of Attributes'	NOA (0, 30)	0
,Number Of Operations'	NOO (0, 50)	2
,Number Of Members' Attribute + Methoden	NOM = NOA + NOO	2
,Number Of Constructors'	NOCON (0, 5)	1
,Number Of Overridden Methods'	NOOM (0, 10)	2
,Percentage of Private Members'	PPrivM	0
,Percentage of Protected Members'	PProtM (0, 10)	0
,Percentage of Public Members'	PPubM	100
,Weighted Methods Per Class'	WMPC1 (0, 30)	
Attribut		
,Attribute Complexity'	AC	0
Methoden		
,Maximum Number Of Parameters'	MNOP (0, 4)	1
,Cyclomatic Complexity'	CC	
Kommentare		
,Number Of Comments'	NOC	9
,True Comment Ratio'	TCR (5, 400)	21

Tabelle 16 „ausgewählte Metriken der Klasse TLoadPointCmd“ (Quelle: Together[®], Version 6.0)

IV.6 Klasse TMoveToPointCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TMoveToPointCmd können erzeugt werden.



IV.6.1 Methoden

► **TMoveToPointCmd(TCmdTag)** **PUBLIC**

Abhängig vom ersten *Kommandoparameter* in den, als Parameter übergebenen, *-informationen* wird eine Absolutposition des ausgewählten Antriebs angefahren.

erster Kommandoparameter	anzufahrende Absolutposition	
Peak	TSteering::dPeakPoint	siehe TLoadPointCmd
Start	TSteering::dStartPoint	
LastGoal	TSteering::dGoalDistance	spart TLoadPointCmd ein
Result	TSteering::dCalcResult	siehe TCalculateCmd
Relative	<aktuelle Antriebsposition> + <dritter <i>Kommandoparameter</i> >	
sonst	<dritter <i>Kommandoparameter</i> >	

Tabelle 17 „Möglichkeiten bei der Positionierung von TMoveToPointCmd“ (Quelle: selbst)

Anschließend ist das Kommando beendet, eStep wird CReady gesetzt.

► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**

überschrieben um Statusinformationen zurückzugeben; Rückgabewert stets TRUE

► **virtual TCode ReadyStep(void)** **PUBLIC ENTF**

eStep wird CReady gesetzt und CReady zurückgegeben

liefert dieselben Ergebnisse wie TCmd; AUSKOMMENTIERT

IV.6.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
‚Lines Of Code‘ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	62
‚LOC of Implementation‘	LOCI	52
‚LOC of Declaration‘	LOCD	10
‚Number Of Attributes‘	NOA (0, 30)	0
‚Number Of Operations‘	NOO (0, 50)	2
‚Number Of Members‘ Attribute + Methoden	NOM = NOA + NOO	2
‚Number Of Constructors‘	NOCON (0, 5)	1
‚Number Of Overridden Methods‘	NOOM (0, 10)	2
‚Percentage of Private Members‘	PPrivM	0
‚Percentage of Protected Members‘	PProtM (0, 10)	0
‚Percentage of Public Members‘	PPubM	100
‚Weighted Methods Per Class‘	WMPC1 (0, 30)	
Attribute		
‚Attribute Complexity‘	AC	0
Methoden		
‚Maximum Number Of Parameters‘	MNOP (0, 4)	1
‚Cyclomatic Complexity‘	CC	
Kommentare		
‚Number Of Comments‘	NOC	12
‚True Comment Ratio‘	TCR (5, 400)	19

Tabelle 18 „ausgewählte Metriken der Klasse TMoveToPointCmd“ (Quelle: Together®, Version 6.0)



IV.7 Klasse TChooseDetectorCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TChooseDeviceCmd können erzeugt werden.

IV.7.1 Methoden

► **TChooseDetectorCmd(TCmdTag)** **PUBLIC**

wählt den Detektor (der als erster *Kommandoparameter* in den *-informationen* übergebenen wird) aus (gültige Werte: Index in `lpDList`) – bei Misserfolg wird `nFailureId` 1 gesetzt. Dann werden `fExposureTime`, `dwExposureCounts` und `fFailure` des ausgewählten Detektors gesetzt. Die Werte sind (durch je ein Leerzeichen voneinander getrennt, also `<float> <DWORD> <float>`) im dritten *Kommandoparameter* gespeichert. Anschließend ist das Kommando beendet, `eStep` wird `CReady` gesetzt.

Wenn der Detektor nicht erfolgreich ausgewählt werden konnte, werden die Parameter – des falschen (aktuellen) Detektors – geändert. KORRIGIERT

► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**

überschrieben um Fehlerinformationen (`nFailureId != 0`) oder Statusinformationen zurückzugeben; Rückgabewert stets `TRUE`

IV.7.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	52
„LOC of Implementation“	LOCI	42
„LOC of Declaration“	LOCD	10
„Number Of Attributes“	NOA (0, 30)	0
„Number Of Operations“	NOO (0, 50)	2
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	2
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	2
„Percentage of Private Members“	PPrivM	0
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	100
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribut		
„Attribute Complexity“	AC	0
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	8
„True Comment Ratio“	TCR (5, 400)	16

Tabelle 19 „ausgewählte Metriken der Klasse TChooseDetectorCmd“ (Quelle: Together[®], Version 6.0)



IV.8 Klasse TGotoIntensityCmd

Deklaration : M_STEERH.CPP

Implementation : M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TGotoIntensityCmd können erzeugt werden.

Dieses Kommando sucht eine Intensität⁹ des ausgewählten Detektors, die durch Bewegung des aktuellen Antriebs angefahren wird. Der erste *Kommandoparameter* regelt ob der Antrieb bevorzugt rückwärts (SmallSide) oder vorwärts (sonst) bewegt werden soll, solange die gesuchte Intensität nicht erreicht ist. Der zweite *Kommandoparameter* bestimmt wie die Antriebsposition zu berechnen und anzufahren ist, wenn an der gesuchten Intensität vorbeigefahren wurde (gültige Werte siehe nReadyAction).

IV.8.1 Attribute

► **int nMotor** **PRIVATE**

Index des, zu Beginn des Kommandos, ausgewählten Antriebs (siehe [1]: mlGetAxis); im Konstruktor initialisiert

► **BOOL bSmallAngleSide** **PRIVATE**

gibt an, ob der erste *Kommandoparameter* SmallSide entspricht; bei TRUE wird der Antrieb in Methode FirstStep rückwärts (sonst vorwärts) bewegt; im Konstruktor initialisiert

Inkonsistenz: wird komplementär zu TGotoPeakCmd verwendet

► **float fSearchIntensity** **PRIVATE**

entspricht der anzusteuernenden Intensität; im Konstruktor als Produkt aus TSteering::dPeakIntensity und dem dritten *Kommandoparameter* berechnet

► **static const UINT nLifo= 3** **PRIVATE** **GEÄ**

bestimmt wie viele Elemente maximal in dDistance und fIntensity enthalten sein sollen; im Konstruktor mit 3 initialisiert (2 würde völlig ausreichen)

sollte unbedingt const sein, um Schreibzugriff zu verhindern (kann static sein); KORRIGIERT

► **double* dDistance** **PRIVATE**

ist eine im Konstruktor dynamisch erzeugte Liste mit nLifo Elementen; Sie dient als „History“ der zuletzt angefahrenen Absolutposition des ausgewählten Antriebs. bei Methode ControlStep wird der aktuelle Wert (dDistance[0]) mit dem letzten Wert verglichen (dDistance[1])

► **float* fIntensity** **PRIVATE**

wie dDistance, jedoch für die Intensitätswerte des verwendeten Detektors

► **TCPParam nReadyAction** **PRIVATE**

bestimmt, wie die endgültige Absolutposition des Antriebs in Methode ControlStep zu berechnen ist; im Konstruktor mit dem Wert des zweiten *Kommandoparameters* initialisiert; gültige Werte

- BackMove (anfahren der zuletzt angefahrenen Absolutposition) und
- Interpolation (Interpolation zwischen aktuellen und letzten Absolutposition und Intensitätswerten)

⁹ Vielfaches aus Peak-Intensität und drittem Kommandoparameter (darf nur float-Werte kleiner gleich 1 annehmen, sonst kommt es zur Endlosschleife)

IV.8.2 Methoden

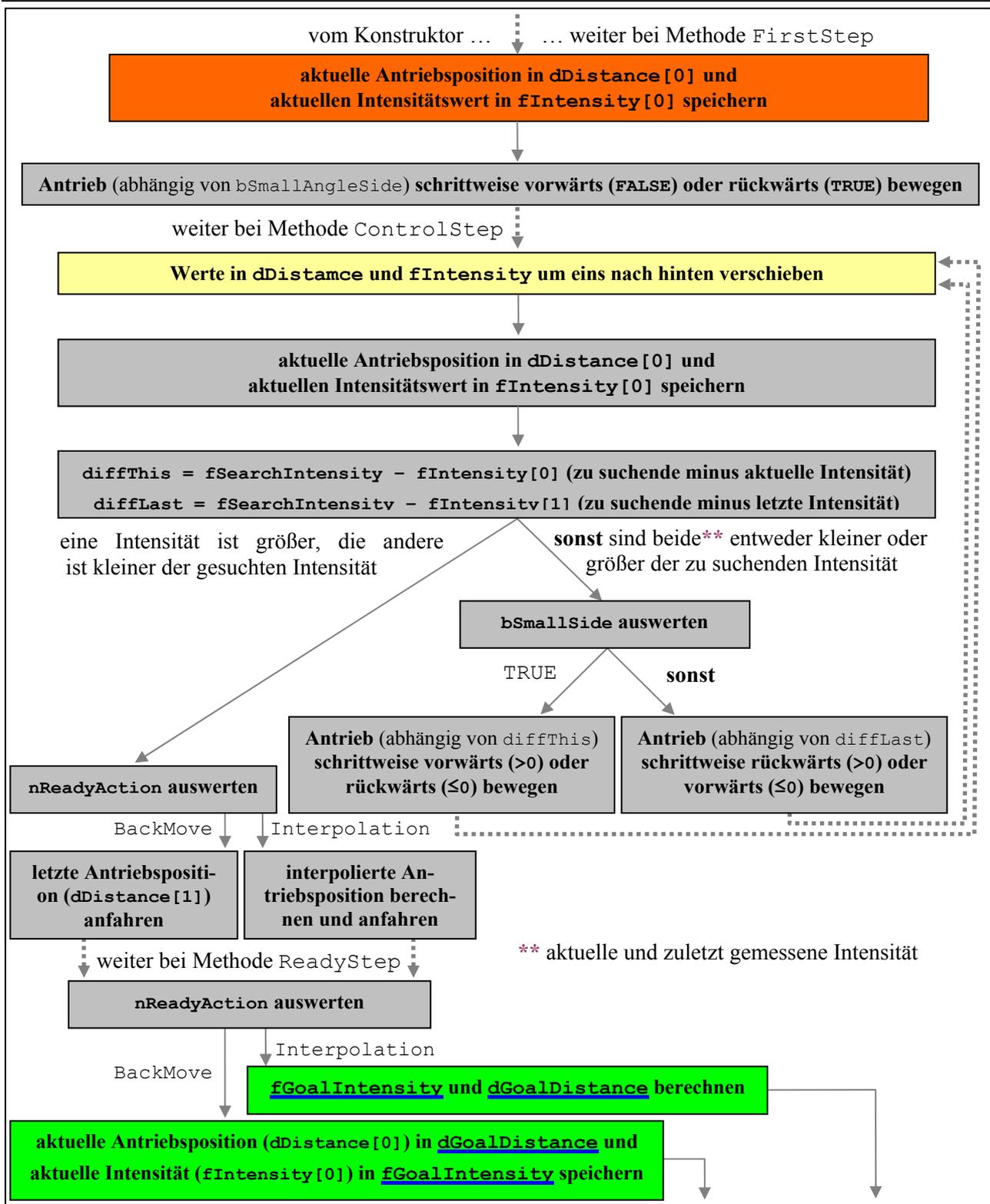


Abbildung 3 „Schematische Darstellung der Abläufe bei TGotoIntensityCmd“ (Quelle: selbst)

blau unterstrichen sind Attribute des Steering-Objekts; punktierte Pfeile bedeuten Kommando-Pausierung



- `TGotoIntensityCmd(TCmdTag)` **PUBLIC**
erzeugt `dDistance` und `fIntensity` dynamisch und initialisiert jedes, der unter [IV.8.1 Attribute](#) genannten, Attribute
- `~TGotoIntensityCmd(void)` **PUBLIC**
gibt die dynamisch erzeugten Listen `dDistance` und `fIntensity` wieder frei
- `virtual TCCode FirstStep(void)` **PUBLIC**
aktuelle Absolutposition und Intensitätswert in `dDistance[0]` und `fIntensity[0]` speichern; abhängig von `bSmallAngleSide` wird der Antrieb entweder um einen Schritt rückwärts (TRUE) oder vorwärts (FALSE) bewegt. nach einer Pausierung der Kommandoverarbeitung wird in Methode `ControlStep` fortgesetzt
- `virtual TCCode ControlStep(void)` **PUBLIC**
Wert in `dDistance` und `fIntensity` um eins nach hinten verschieben („History“) und die aktuelle Absolutposition und Intensitätswert in `dDistance[0]` und `fIntensity[0]` speichern;
`diffThis = fSearchIntensity - fIntensity[0]` und
`diffLast = fSearchIntensity - fIntensity[1]`
- `diffThis · diffLast < 0` (eine Intensität ist größer, die andere kleiner der gesuchten Intensität)
In Abhängigkeit von `nReadyAction` wird eine Antriebsposition interpoliert und angefahren (Interpolation) oder die zuletzt verwendete angefahren (BackMove) berechnet und angefahren. nach einer Pausierung der Kommandoverarbeitung wird in Methode `ReadyStep` fortgesetzt.
Sollte hier nicht auch der Fall `diffThis · diffLast == 0` (aktuelle oder letzte Intensität stimmt mit der gesuchten Intensität überein) mit eingeschlossen werden?
 - sonst (entweder sind beide¹⁰ kleiner oder größer der gesuchten Intensität)
In beiden Fällen wird nach der Pausierung der Kommandoverarbeitung wieder Methode `ControlStep` aufgerufen.
 - `bSmallAngleSide == TRUE`
wenn `diffThis > 0` (aktuelle ist kleiner der gesuchten Intensität), dann Antrieb einen Schritt vorwärts (sonst rückwärts bewegt)
 - sonst
wenn `diffLast > 0` (letzte ist kleiner der gesuchten Intensität), dann Antrieb einen Schritt rückwärts (sonst vorwärts bewegen)
Warum wird hier nicht `diffThis` ausgewertet? Diese Abfrage ist im Quelltext als vermutliche Fehlerquelle gekennzeichnet.
- In jedem Fall wird `CRecall` zurückgegebenen
- `virtual TCCode ReadyStep(void)` **PUBLIC**
wenn `nReadyAction == Interpolation` ist, wird `TSteering::dGoalDistance` und `TSteering::fGoalIntensity` berechnet; bei `BackMove` wird die aktuelle Absolutposition in `TSteering::dGoalDistance` und die aktuelle Intensität in `TSteering::fGoalIntensity` gespeichert; abschließend kennzeichnet `eStep = CReady`, dass die Kommandoverarbeitung erfolgreich beendet wurde; gibt `CReady` zurück

¹⁰ die aktuelle und letzte Intensität



► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**
 überschrieben um Fehlerinformationen (`nFailureId != 0`) oder Statusinformationen zurückzugeben:

eStep	Rückgabe im ersten Parameter
CReady	"GotoIntensity Ready"
CFirstStep	"Ziel-Intensität <i>" (wobei <i> = fSearchIntensity ist)
CControlStep	"D:<d> I:<i>" (wobei <d> = dDistance[0] und <i> = fIntensity[0] ist)
sonst	0

Tabelle 20 „Möglichkeiten der Textrückgabe bei `TGotoIntensityCmd::GetShowData`“ (Quelle: selbst)
 Rückgabewert stets TRUE

IV.8.3 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	70
„LOC of Implementation“	LOCI	35
„LOC of Declaration“	LOCD	35
„Number Of Attributes“	NOA (0, 30)	7
„Number Of Operations“	NOO (0, 50)	5
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	12
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	5
„Percentage of Private Members“	PPrivM	50
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	50
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	48
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	47
„True Comment Ratio“	TCR (5, 400)	29

Tabelle 21 „ausgewählte Metriken der Klasse `TGotoIntensityCmd`“ (Quelle: Together[®], Version 6.0)

IV.9 Klasse TGotoPeakCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von `TGotoPeakCmd` können erzeugt werden.



IV.9.1 Attribute

- **int nMotor** **PRIVATE**
Index des ausgewählten Antriebs (siehe [1]: mlGetAxis); im Konstruktor mit aktuell ausgewähltem Antrieb initialisiert
- **BOOL bSmallAngleSide** **PRIVATE**
gibt an, ob der erste *Kommandoparameter* SmallSide entspricht; bei TRUE wird der Antrieb in Methode FirstStep vorwärts (sonst nach rückwärts) bewegt; im Konstruktor initialisiert
- **BOOL bFlankDetected** **PRIVATE**
gibt an, dass eine Antriebsposition mit "hinreichender Genauigkeit zum Peak" erreicht wurde; list dies der Fall, wird die Bewegungsrichtung geändert und der ausgewählte Antrieb wird auf den Peak eingependelt. im Konstruktor mit FALSE initialisiert
- **TCPParam nReadyAction** **PRIVATE**
bestimmt, wie TSteering::dGoalDistance, TSteering::dPeakPoint und TSteering::fPeakIntensity in Methode ReadyStep zu berechnen sind (nReadyAction == Interpolation) bzw. dass der Antrieb die letzte Position wieder anfahren soll (BackMove)
- **static UINT nLifo= 3** **PRIVATE** **GEÄ**
bestimmt wie viele Elemente in dDistance und fIntensity maximal enthalten sein sollen; im Konstruktor mit 3 initialisiert
sollte unbedingt const sein, um fälschlichen Schreibzugriff zu verhindern (kann static sein); KORRIGIERT
- **double* dDistance** **PRIVATE**
ist eine im Konstruktor dynamisch erzeugte Liste mit nLifo Elementen; Sie dient als „History“ der zuletzt angefahrenen Absolutposition des ausgewählten Antriebs.
- **float* fIntensity** **PRIVATE**
wie dDistance, jedoch für die Intensitätswerte des verwendeten Detektors
- **BOOL bBackMoveActive** **PRIVATE**
gibt an, dass beim nächsten Aufruf von Methode ReadyStep Berechnungen durchzuführen sind, danach ist die Kommandoverarbeitung erfolgreich beendet
nicht initialisiert, wenn bBackMove uninitialized TRUE annimmt, wird trotz nReadyAction == Interpolation die Berechnung immer für BackMove durchgeführt, der Antrieb wurde aber nicht auf die für BackMove gewünschte, letzte Absolutposition gefahren.
- **float fSearchIntensity** **PRIVATE**
nur ein Lesezugriff zur Berechnung von TSteering::dGoalDistance; kein Schreibzugriff
- **BOOL bDynamicStepWidth** **PRIVATE** **ENTF**
► **BOOL bDirectionChanged** **PRIVATE** **ENTF**
toter Code; nur ein Schreibzugriff beim Initialisieren im Konstruktor; AUSKOMMENTIERT

IV.9.2 Methoden

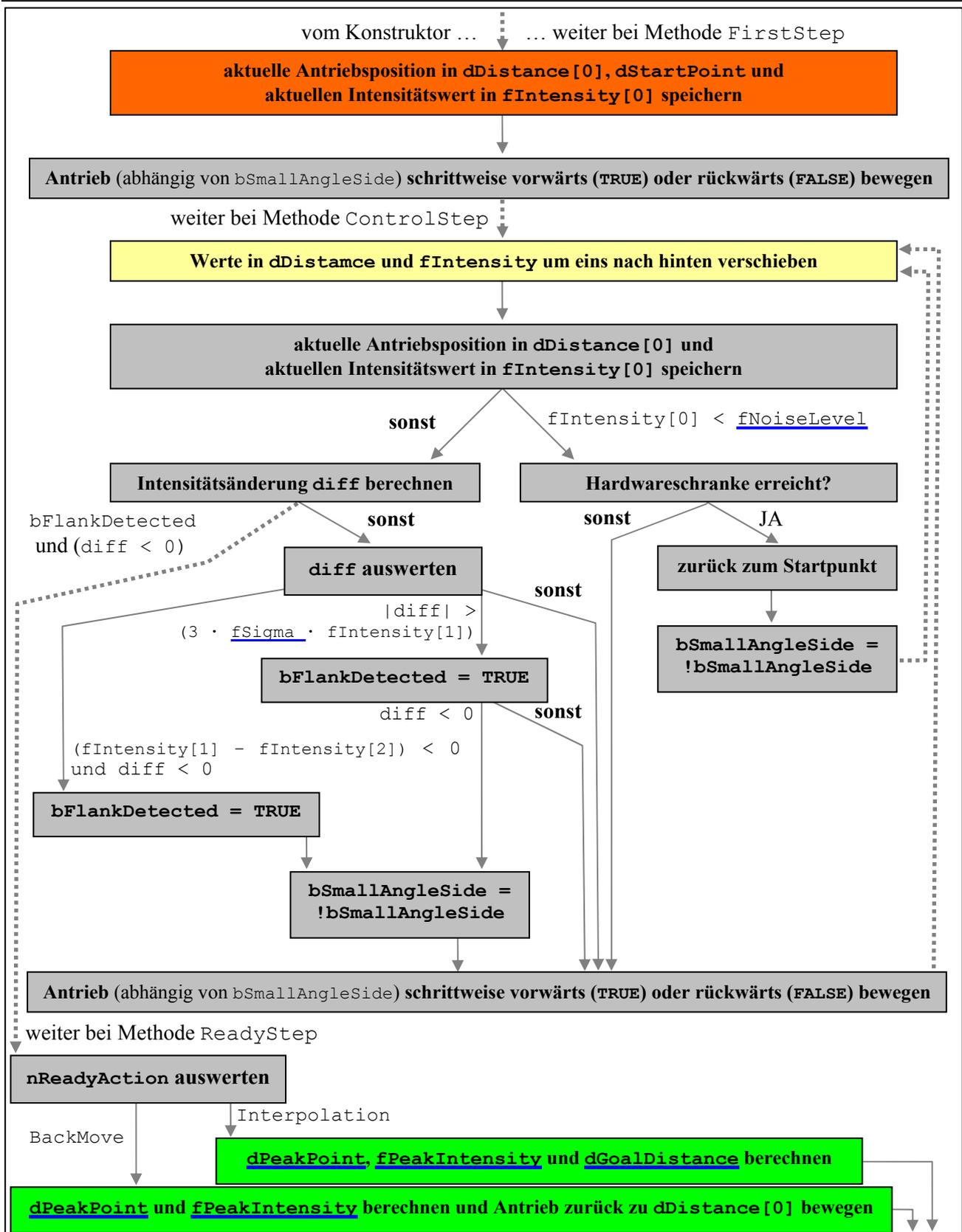


Abbildung 4 „Schematische Darstellung der Abläufe bei TGotoPeakCmd“ (Quelle: selbst)

blau unterstrichen sind Attribute des Steering-Objekts; punktierte Pfeile bedeuten Kommando-Pausierung



► **TGotoPeakCmd(TCmdTag)** **PUBLIC**
erzeugt `dDistance` und `fIntensity` dynamisch und initialisiert einige (unter [IV.9.1 Attribute](#) genannte) Attribute

► **~TGotoPeakCmd()** **PUBLIC**
gibt die dynamisch erzeugten Listen `dDistance` und `fIntensity` wieder frei

► **virtual TCode FirstStep(void)** **PUBLIC**
aktuelle Absolutposition in `dDistance[0]` und `TSteering::dStartPoint` und aktuelle Intensitätswert in `fIntensity[0]` speichern; Abhängig von `bSmallAngleSide` wird der Antrieb entweder um einen Schritt vorwärts (TRUE) oder rückwärts (FALSE) bewegt. nach einer Pausierung der Kommandoverarbeitung wird in Methode `ControlStep` fortgesetzt

► **virtual TCode ControlStep(void)** **PUBLIC**
Wert in `dDistance` und `fIntensity` um eins nach hinten verschieben („History“) und die aktuelle Absolutposition und Intensitätswert in `dDistance[0]` und `fIntensity[0]` speichern

`diff = fIntensity[0] - fIntensity[1]` Intensitätsänderung seit letzter Messung

- aktuelle Intensität ist unter dem Rausch-Level (`TSteering::fNoiseLevel`)
In beiden Fällen wird nach einer Pausierung der Kommandoverarbeitung wieder Methode `ControlStep` aufgerufen.
 - Hardwareschranke des ausgewählten Antriebs erreicht, dann bewege den Antrieb zurück zum Startpunkt (`TSteering::dStartPoint`), ändere die bevorzugte Bewegungsrichtung (`bSmallAngleSide`) und starte nach einer Pausierung wieder in Methode `ControlStep`
 - sonst wird der Antrieb wie in Methode `ReadStep` schrittweise auf den Peak zu bewegt
- `bFlankDetected == TRUE` und `diff < 0` **Peak gefunden**
nach einer Pausierung der Kommandoausführung weiter bei Methode `ReadyStep`
- `| diff | > 3 * TSteering::fSigma * fIntensity[1]` **große Intensitätsänderung**
`bFankDetected TRUE` setzen; wenn die aktuelle kleiner ist als zuletzt gemessene Intensität, dann ändere die bevorzugte Bewegungsrichtung (`bSmallAngleSide`); bewege den Antrieb schrittweise in Richtung Peak (wie Methode `FirstStep`) und starte nach einer Pausierung wieder bei Methode `ControlStep`
- `diff < 0` und `(fIntensity[1] - fIntensity[2]) < 0`
(Intensität hat während der letzten beiden Messungen abgenommen)
`bFankDetected TRUE` setzen; ändere die bevorzugte Bewegungsrichtung (`bSmallAngleSide`); bewege den Antrieb schrittweise in Richtung Peak (wie Methode `FirstStep`) und starte nach einer Pausierung wieder bei Methode `ControlStep`
- sonst bewege den Antrieb schrittweise in Richtung Peak (wie in Methode `FirstStep`) und starte nach einer Pausierung wieder bei Methode `ControlStep`

In jedem Fall wird `CRcall` zurückgegeben



► **virtual TCCode ReadyStep(void) PUBLIC**

TSteering::dPeakPoint und TSteering::fPeakIntensity werden je nach nReadyAction unterschiedlich berechnet; wenn nReadyAction == Interpolation ist, wird auch TSteering::dGoalDistance berechnet, bei BackMove wird der Antrieb zur letzten Antriebsposition gefahren; abschließend kennzeichnet eStep = CReady, dass die Kommandoverarbeitung erfolgreich beendet wurde; gibt CReady zurück

Verwendung von zwei nicht initialisierten Attributen: bBackMoveAction und fSearchIntensity.

► **virtual BOOL GetShowData(LPSTR) PUBLIC**

überschrieben um Fehlerinformationen (nFailureId != 0) oder Statusinformationen zurückzugeben:

eStep	Rückgabe im ersten Parameter
CReady	"GotoPeak Ready"
CControlStep	"D:<d> I:<i>" (wobei <d> = dDistance[0] und <i> = fIntensity[0] ist)
sonst	0

Tabelle 22 „Möglichkeiten der Textrückgabe bei TGotoPeakCmd::GetShowData“ (Quelle: selbst)

Rückgabewert stets TRUE

IV.9.3 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
‚Lines Of Code‘ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	190
‚LOC of Implementation‘	LOCI	153
‚LOC of Declaration‘	LOCD	37
‚Number Of Attributes‘	NOA (0, 30)	9
‚Number Of Operations‘	NOO (0, 50)	5
‚Number Of Members‘ Attribute + Methoden	NOM = NOA + NOO	14
‚Number Of Constructors‘	NOCON (0, 5)	1
‚Number Of Overridden Methods‘	NOOM (0, 10)	5
‚Percentage of Private Members‘	PPrivM	56
‚Percentage of Protected Members‘	PProtM (0, 10)	0
‚Percentage of Public Members‘	PPubM	44
‚Weighted Methods Per Class‘	WMPC1 (0, 30)	
Attribute		
‚Attribute Complexity‘	AC	66
Methoden		
‚Maximum Number Of Parameters‘	MNOP (0, 4)	1
‚Cyclomatic Complexity‘	CC	
Kommentare		
‚Number Of Comments‘	NOC	59
‚True Comment Ratio‘	TCR (5, 400)	40

Tabelle 23 „ausgewählte Metriken der Klasse TGotoPeakCmd“ (Quelle: Together®, Version 6.0)



IV.10 Klasse TShowValueCmd

Deklaration : M_STEERH.CPP

Implementation : M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TShowValueCmd können erzeugt werden.

IV.10.1 Methoden

► **TShowValueCmd(TCmdTag) PUBLIC**

Abhängig vom ersten *Kommandoparameter* in den, als Parameter übergebenen, *-informationen* wird ein Meldungsfenster angezeigt. Anschließend ist das Kommando beendet, eStep wird CReady gesetzt.

erster Kommandoparameter	angezeigter Text
Hwb	"Halbwertsbreite : <h> arcsec", wobei <h> = TSteering::fHwb
Peak	"Peak (D,I) : (<d>,<i>)", wobei <d> = TSteering::dPeakPoint und <i> = TSteering::fPeakIntensity
Start	"Startposition : <s>", wobei <s> = TSteering::dStartPoint

Tabelle 24 „Möglichkeiten bei der Positionierung von TShowValueCmd“ (Quelle: selbst)

► **virtual BOOL GetShowData(LPSTR) PUBLIC**

überschrieben um 0 zurückzugeben; Rückgabewert stets TRUE

IV.10.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	41
„LOC of Implementation“	LOCI	31
„LOC of Declaration“	LOCD	10
„Number Of Attributes“	NOA (0, 30)	0
„Number Of Operations“	NOO (0, 50)	2
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	2
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	2
„Percentage of Private Members“	PPrivM	0
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	100
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	0
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	7
„True Comment Ratio“	TCR (5, 400)	18

Tabelle 25 „ausgewählte Metriken der Klasse TShowValueCmd“ (Quelle: Together®, Version 6.0)



IV.11 Klasse TCalculateCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TCalculateCmd können erzeugt werden.

IV.11.1 Methoden

► **TCalculateCmd(TCmdTag)** **PUBLIC**

Abhängig vom ersten *Kommandoparameter* in den, als Parameter übergebenen, *-informationen* werden verschiedene Werte berechnet und in TSteering::dCalcResult oder TSteering::fHwb gespeichert.

erster Kommandoparameter	welches Attribut wird geschrieben (Attribute von TSteering)
Difference	$dCalcResult = dCalcArg[2] - dCalcArg[1]$
Opposite	$dCalcResult = 2 \cdot dPeakPoint - dCalcArg[1]$
Hwb	$fHwb = dCalcArg[2] - dCalcArg[1]$ wird entsprechend der Nutzereinheit (des ausgewählten Antriebs) weiter umgerechnet
Middle	$dCalcResult = (dCalcArg[1] + dCalcArg[2]) / 2$

Tabelle 26 „Möglichkeiten bei der Berechnung von TCalculateCmd“ (Quelle: selbst)

Anschließend ist das Kommando beendet, eStep wird CReady zugewiesen.

► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**

überschrieben um Statusinformationen zurückzugeben; Rückgabewert stets TRUE

**IV.11.2 Bewertung**

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	50
„LOC of Implementation“	LOCI	40
„LOC of Declaration“	LOCD	10
„Number Of Attributes“	NOA (0, 30)	0
„Number Of Operations“	NOO (0, 50)	2
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	2
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	2
„Percentage of Private Members“	PPrivM	0
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	100
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	0
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	7
„True Comment Ratio“	TCR (5, 400)	14

Tabelle 27 „ausgewählte Metriken der Klasse TCalculateCmd“ (Quelle: Together[®], Version 6.0)**IV.12 Klasse TInquireCmd**

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TInquireCmd können erzeugt werden.

Die Klasse ist (nach Quelle: [6]) toter Code. AUSKOMMENTIERT**IV.12.1 Methoden**

```

▶ TInquireCmd( TCmdTag ) PUBLIC ENTFF
setzt bNoMeasure auf TRUE (Detektormessung sollen nicht durchgeführt werden) und eStep auf
CReady

```

```

▶ virtual BOOL GetShowData( LPSTR ) PUBLIC ENTFF
überschrieben um 0 zurückzugeben; Rückgabewert stets TRUE

```

**IV.12.2 Bewertung**

Metrik		Kennung (min,max)	Wert
Klasse			
,Lines Of Code' inkl. Leer- und Kommentarzeilen		LOC (0, 1000)	T O T E R C O D E
,LOC of Implementation'		LOCI	
,LOC of Declaration'		LOCD	
,Number Of Attributes'		NOA (0, 30)	
,Number Of Operations'		NOO (0, 50)	
,Number Of Members' Attribute + Methoden		NOM = NOA + NOO	
,Number Of Constructors'		NOCON (0, 5)	
,Number Of Overridden Methods'		NOOM (0, 10)	
,Percentage of Private Members'		PPrivM	
,Percentage of Protected Members'		PProtM (0, 10)	
,Percentage of Public Members'		PPubM	
,Weighted Methods Per Class'		WMPC1 (0, 30)	
Attribute			
,Attribute Complexity'		AC	
Methoden			
,Maximum Number Of Parameters'		MNOP (0, 4)	
,Cyclomatic Complexity'		CC	
Kommentare			
,Number Of Comments'		NOC	
,True Comment Ratio'		TCR (5, 400)	

Tabelle 28 „ausgewählte Metriken der Klasse TInquireCmd“ (Quelle: Together[®], Version 6.0)**IV.13 Klasse TControlFlankCmd**

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TControlFlankCmd können erzeugt werden.

IV.13.1 Attribute

▶ <code>int</code>	<code>nMotor</code>	<code>PRIVATE</code>
▶ <code>float</code>	<code>fControlRange</code>	<code>PRIVATE</code>
▶ <code>float</code>	<code>fStartIntensity</code>	<code>PRIVATE</code>
▶ <code>float</code>	<code>fSearchIntensity</code>	<code>PRIVATE</code>
▶ <code>float</code>	<code>fMinIntensity</code>	<code>PRIVATE</code>
▶ <code>float</code>	<code>fMaxIntensity</code>	<code>PRIVATE</code>
▶ <code>float</code>	<code>fIntensityRange</code>	<code>PRIVATE</code>
▶ <code>BOOL</code>	<code>bSmallAngleSide</code>	<code>PRIVATE</code>



▶ **static const UINT nLifo= 4** **PRIVATE** **GEÄ**
bestimmt wie viele Elemente maximal in dDistance und fIntensity enthalten sein sollen; im Kon-
struktor mit 4 initialisiert (2 würde völlig ausreichen)

**sollte unbedingt const sein, um fälschlichen Schreibzugriff zu verhindern (kann static sein);
KORRIGIERT**

▶ **double* dDistance** **PRIVATE**

▶ **float* fIntensity** **PRIVATE**

IV.13.2 Methoden

▶ **TControlFlankCmd(TCmdTag)** **PUBLIC**

▶ **~TControlFlankCmd()** **PUBLIC**

▶ **virtual TCCode FirstStep(void)** **PUBLIC**

▶ **virtual TCCode ControlStep(void)** **PUBLIC**

▶ **virtual TCCode ReadyStep(void)** **PUBLIC**

▶ **virtual BOOL GetShowData(LPSTR)** **PUBLIC**

überschrieben um Statusinformationen zurückzugeben; Rückgabewert stets TRUE

**IV.13.3 Bewertung**

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	198
„LOC of Implementation“	LOCI	162
„LOC of Declaration“	LOCD	36
„Number Of Attributes“	NOA (0, 30)	11
„Number Of Operations“	NOO (0, 50)	5
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	16
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	5
„Percentage of Private Members“	PPrivM	61
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	39
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	49
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	84
„True Comment Ratio“	TCR (5, 400)	65

Tabelle 29 „ausgewählte Metriken der Klasse TControlFlankCmd“ (Quelle: Together[®], Version 6.0)**IV.14 Klasse TSetupScanCmd**

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TSetupScanCmd können erzeugt werden.

IV.14.1 Attribute► **TScan* ScanW** **PRIVATE****IV.14.2 Methoden**► **TSetupScanCmd(TCmdTag)** **PUBLIC**► **virtual BOOL GetShowData(LPSTR)** **PUBLIC**überschrieben um Fehlerinformationen (nFailureId == 1) oder Statusinformationen zurückzugeben;
Rückgabewert stets TRUE

**IV.14.3 Bewertung**

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	61
„LOC of Implementation“	LOCI	48
„LOC of Declaration“	LOCD	13
„Number Of Attributes“	NOA (0, 30)	1
„Number Of Operations“	NOO (0, 50)	2
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	3
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	2
„Percentage of Private Members“	PPrivM	25
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	75
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	9
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	8
„True Comment Ratio“	TCR (5, 400)	14

Tabelle 30 „ausgewählte Metriken der Klasse TSetupScanCmd“ (Quelle: Together[®], Version 6.0)**IV.15 Klasse TSetFileNameCmd**

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TSetFileNameCmd können erzeugt werden.

IV.15.1 Attribute▶ **TScan* ScanW** **PRIVATE**▶ **TAreaScan* AreaScanW** **PRIVATE****IV.15.2 Methoden**▶ **TSetFileNameCmd(TCmdTag)** **PUBLIC**▶ **virtual BOOL GetShowData(LPSTR)** **PUBLIC**überschrieben um Fehlerinformationen (nFailureId == 1) oder Statusinformationen zurückzugeben;
Rückgabewert stets TRUE

**IV.15.3 Bewertung**

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	53
„LOC of Implementation“	LOCI	39
„LOC of Declaration“	LOCD	14
„Number Of Attributes“	NOA (0, 30)	2
„Number Of Operations“	NOO (0, 50)	2
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	4
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	2
„Percentage of Private Members“	PPrivM	40
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	60
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	18
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	7
„True Comment Ratio“	TCR (5, 400)	13

Tabelle 31 „ausgewählte Metriken der Klasse TSetFileNameCmd“ (Quelle: Together[®], Version 6.0)**IV.16 Klasse TSaveDataCmd**

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TSaveDataCmd können erzeugt werden.

IV.16.1 Attribute

- ▶ **TScan* ScanW** **PRIVATE**
- ▶ **TAreaScan* AreaScanW** **PRIVATE**

IV.16.2 Methoden

- ▶ **TSaveDataCmd (TCmdTag)** **PUBLIC**
- ▶ **virtual BOOL GetShowData (LPSTR)** **PUBLIC**
 überschrieben um Fehlerinformationen (nFailureId != 0) oder Statusinformationen zurückzugeben;
 Rückgabewert stets TRUE

**IV.16.3 Bewertung**

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	55
„LOC of Implementation“	LOCI	41
„LOC of Declaration“	LOCD	14
„Number Of Attributes“	NOA (0, 30)	2
„Number Of Operations“	NOO (0, 50)	2
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	4
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	2
„Percentage of Private Members“	PPrivM	40
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	60
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	18
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	7
„True Comment Ratio“	TCR (5, 400)	13

Tabelle 32 „ausgewählte Metriken der Klasse TSaveDataCmd“ (Quelle: Together[®], Version 6.0)

IV.17 Klasse TScanCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TScanCmd können erzeugt werden.

IV.17.1 Attribute, Methoden

Die für die Diffraktometrie relevanten Kommandos wurde bereits einem Reverse-Engineering Prozess unterzogen – Informationen findet man unter [\[3\]](#).

► **DWORD dwStartTimeTicks**

PRIVATE

NEU

Anfangszeit des Scans; benutzt um Restzeit zu berechnen



IV.17.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	503
„LOC of Implementation“	LOCI	355
„LOC of Declaration“	LOCD	148
„Number Of Attributes“	NOA (0, 30)	32
„Number Of Operations“	NOO (0, 50)	5
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	37
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	5
„Percentage of Private Members“	PPrivM	84
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	16
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	142
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	179
„True Comment Ratio“	TCR (5, 400)	53

Tabelle 33 „ausgewählte Metriken der Klasse TScanCmd“ (Quelle: Together[®], Version 6.0)

IV.18 Klasse TAreaScanCmd

Deklaration : M_STEERH.CPP

Implementation: M_STEERG.H

Es gibt keine abstrakten Methoden, d.h. Objekte von TAreaScanCmd können erzeugt werden.

IV.18.1 Attribute, Methoden

Die für die Diffraktometrie relevanten Kommandos wurde bereits einem Reverse-Engineering Prozess unterzogen – Informationen findet man unter [\[3\]](#).



IV.18.2 Bewertung

Metrik	Kennung (min,max)	Wert
Klasse		
„Lines Of Code“ inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	226
„LOC of Implementation“	LOCI	157
„LOC of Declaration“	LOCD	69
„Number Of Attributes“	NOA (0, 30)	18
„Number Of Operations“	NOO (0, 50)	5
„Number Of Members“ Attribute + Methoden	NOM = NOA + NOO	23
„Number Of Constructors“	NOCON (0, 5)	1
„Number Of Overridden Methods“	NOOM (0, 10)	5
„Percentage of Private Members“	PPrivM	75
„Percentage of Protected Members“	PProtM (0, 10)	0
„Percentage of Public Members“	PPubM	25
„Weighted Methods Per Class“	WMPC1 (0, 30)	
Attribute		
„Attribute Complexity“	AC	37
Methoden		
„Maximum Number Of Parameters“	MNOP (0, 4)	1
„Cyclomatic Complexity“	CC	
Kommentare		
„Number Of Comments“	NOC	81
„True Comment Ratio“	TCR (5, 400)	52

Tabelle 34 „ausgewählte Metriken der Klasse TAreaScanCmd“ (Quelle: Together[®], Version 6.0)

V Attribute

▶ **TSteering Steering**

GLOBAL

Dieses Objekt kann nur zur Makroverarbeitung benutzt werden. Es dürfen keine weiteren Objekte von TSteering erstellt werden (siehe Konstruktor von TSteering).

▶ **static const UINT _MAXMACROCOUNT= 20**

GLOBAL

NEU

definiert die maximale Anzahl von Makros, kann kein static-const Member in TSteering sein

▶ **TAdjustmentParameter AdjustmentParameter**

GLOBAL

Justageparameter; wird in TSetAdjustmentParam mit den eingegebenen Werten initialisiert; in TAdjustmentExecute wird nur das Element fNoiseLevel verwendet

▶ **DWORD dwStartTimeTicks**

GLOBAL

ENTF

Anfangszeit des Scans; benutzt um Restzeit zu berechnen

wird nur in TScanCmd benutzt; sollte als private Member in TScanCmd aufgenommen werden; KORRIGIERT



- **HGDIOBJ testobjc** GLOBAL ENTFF
Handle zu dem vor SelectObject() ausgewählten Objekt
wird nur in Methode TEditWindow::Paint() geschrieben; sollte dort als lokale Variable verwendet und am Ende des Zeichenvorgangs wieder ausgewählt werden; KORRIGIERT
- **static UINT nEvent** GLOBAL ENTFF
Handle auf einen aktiven Timer in TSteering oder wenn TTSteering-Timer inaktiv ist 0.
nEvent sollte nicht global deklariert sein, sondern als private Member in TSteering aufgenommen werden; KORRIGIERT
- **static HWND hHostWindow** GLOBAL ENTFF
Handle zu dem Fenster, das über die Windowsbotschaft cm_SteeringReady benachrichtigt werden soll, wenn die Makroverarbeitung beendet ist
wird nicht initialisiert, beim Sender der Nachricht wird nicht überprüft, ob hHostWindows != 0 ist; sollte als Member in TSteering aufgenommen werden; KORRIGIERT
- **TDetector *Monitor** GLOBAL ENTFF
Verknüpfung auf einen Detektor für TScanCmd, TAreaScanCmd und TGotoPeakCmd
sollte als Member in TSteering aufgenommen werden; KORRIGIERT
- **TExposureSettings exposure** GLOBAL ENTFF
speichert Detektorparameter vor dem Pausieren der Makroverarbeitung, um diese beim Fortsetzen wiederherstellen zu können
sollte als private Member in TSteering aufgenommen werden; KORRIGIERT
- **static const UINT nAskTicks= 30** GLOBAL ENTFF
Diese Konstante gibt an, wie lange [in Millisekunden] die Kommandoverarbeitung dauern soll
sollte als static-const Member in TSteering aufgenommen werden KORRIGIERT
- **static const int nMaxAskNumber= 200** GLOBAL ENTFF
Diese Konstante gibt an, wie oft die Kommandoverarbeitung maximal pausieren darf.
sollte als static-const Member in TSteering aufgenommen werden.
Nachdem nAskId entfernt wurde, war nMaxNumber toter Code. AUSKOMMENTIERT
- **static int nAskIdx** GLOBAL ENTFF
gibt an, wie oft die Kommandoverarbeitung noch pausieren darf (wird bei jeder Pause dekrementiert) – initialisiert mit nMaxNumber; Der einzige Lesezugriff, nAskId > 0, wurde auskommentiert, damit
toter Code, kann entfernt werden AUSKOMMENTIERT
- **TAdjustmentWindow *AdjWnd** GLOBAL ENTFF
wird nur an einer Stelle beschrieben; AUSKOMMENTIERT
- **BOOL bMonitorUsed** GLOBAL ENTFF
wird nur lokal verwendet; AUSKOMMENTIERT



VI Methoden

► **void CALLBACK RecallSteering(UINT, UINT, DWORD, GLOBAL ENTFF
DWORD, DWORD)**

wird beim Aufruf des Timers gerufen; setzt `nEvent = 0` und kennzeichnet damit, dass der Timer nun inaktiv ist; Anschließend wird die Windows-Botschaft `wm_WakeUpSteering` ans Hauptfenster ‚Steuerprogramm‘ geschickt. Dort wird Methode `Steering->WakeUp` aufgerufen.

sollte als static Member in TSteering aufgenommen werden (CALLBACK-Methoden müssen static sein!) KORRIGIERT

VII Anhang

VII.1 Verwandte Dokumente

- [1] „Beschreibung einer Schnittstelle zur Motorenansteuerung: Das C-Interface des RTK-Steuerungsprogramms“, Studienarbeit von Sebastian Freund und Derrick Hepp
- [2] „Reverse-Engineering der objektorientierten Teile des Subsystems Motorsteuerung“, Version 1.5 von Thomas Kullmann und Günther Reinecker
- [3] „Reverse-Engineering des Subsystems Detektoren des RTK-Steuerprogramms“, November 2000 von Jan Picard, René Harder und Alexander Paschold
- [4] „Der Aufbau von Makros im XRay-Programm“, am 09.09.1999 von Kay Schützler erstellt
- [5] „Pflichtenheft ‚Manuelle Justage‘“, Version 2.1 von Thomas Kullmann und Günther Reinecker
- [6] „Toter Code: Ergebnisse mit SNiFF und McCabe“, am 15.09.1999 von Stefan Lützkendorf erstellt
- [7] „Layoutkonventionen und Steuerelemente“, Version 1.0 von Thomas Kullmann und Günther Reinecker

VII.2 Index

EINZELSCHRITT	3
KOMMANDO	3
KOMMANDOINFORMATION	8
KOMMANDOPARAMETER.....	3
MAKRO.....	3



VII.3 Tabellen

TABELLE 1 „AUFLISTUNG DER ZUM SUBSYSTEM ZUGEHÖRIGEN DATEIEN“ (QUELLE: SELBST).....	3
TABELLE 2 „BEZIEHUNGEN ZWISCHEN BEZEICHNUNG UND EINDEUTIGEM MAKROTYP – GROB-/ KLEINSCHREIBUNG IST ZU BEACHTEN!“ (QUELLE: NACH STANDARD.MAK UND SCAN.MAK).....	5
TABELLE 3 „ATTRIBUTE VON T _{MACROTAG} “ (QUELLE: SELBST).....	6
TABELLE 4 „BEZIEHUNGEN ZWISCHEN BEZEICHNUNG, EINDEUTIGEM KOMMANDOTYP UND BENUTZUNG – GROB-/ KLEINSCHREIBUNG IST ZU BEACHTEN!“ (QUELLE: SELBST).....	7
TABELLE 5 „BEZIEHUNGEN ZWISCHEN BEZEICHNUNG UND EINDEUTIGEM KOMMANDOPARAMETER-TYP – GROB-/ KLEINSCHREIBUNG IST ZU BEACHTEN!“ (QUELLE: SELBST).....	8
TABELLE 6 „ATTRIBUTE VON T _{CMDDTAG} “ (QUELLE: SELBST).....	8
TABELLE 7 „SYMBOLISCHE WERTE VON T _{CCODE} “ (QUELLE: SELBST).....	9
TABELLE 8 „FRIENDS DER KLASSE T _{STEERING} “ (QUELLE: SELBST).....	11
TABELLE 9 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{STEERING} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	23
TABELLE 10 „FRIENDS DER KLASSE T _{CMD} “ (QUELLE: SELBST).....	24
TABELLE 11 „BEDEUTUNG DER WERTE IN DEN KLASSEN, WO T _{FAILURE} BENUTZT WIRD“ (QUELLE: SELBST).....	24
TABELLE 12 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{CMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	27
TABELLE 13 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{CHOOSEAXISCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	28
TABELLE 14 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{SETWIDTHCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	29
TABELLE 15 „MÖGLICHKEITEN BEI DER ARBEITSWEISE VON T _{LOADPOINTCMD} “ (QUELLE: SELBST).....	30
TABELLE 16 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{LOADPOINTCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	30
TABELLE 17 „MÖGLICHKEITEN BEI DER POSITIONIERUNG VON T _{MOVETOPOINTCMD} “ (QUELLE: SELBST).....	31
TABELLE 18 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{MOVETOPOINTCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	31
TABELLE 19 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{CHOOSEDETECTORCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	32
TABELLE 20 „MÖGLICHKEITEN DER TEXTRÜCKGABE BEI T _{GOTOINTENSITYCMD} : GETSHOWDATA“ (QUELLE: SELBST).....	36
TABELLE 21 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{GOTOINTENSITYCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	36
TABELLE 22 „MÖGLICHKEITEN DER TEXTRÜCKGABE BEI T _{GOTOPEAKCMD} : GETSHOWDATA“ (QUELLE: SELBST).....	40
TABELLE 23 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{GOTOPEAKCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	40
TABELLE 24 „MÖGLICHKEITEN BEI DER POSITIONIERUNG VON T _{SHOWVALUECMD} “ (QUELLE: SELBST).....	41
TABELLE 25 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{SHOWVALUECMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	41
TABELLE 26 „MÖGLICHKEITEN BEI DER POSITIONIERUNG VON T _{SHOWVALUECMD} “ (QUELLE: SELBST).....	42
TABELLE 27 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{CALCULATECMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	43
TABELLE 28 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{INQUIRECMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	44
TABELLE 29 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{CONTROLFLANKCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	46
TABELLE 30 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{SETUPSCANCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	47
TABELLE 31 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{SETFILENAMECMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	48
TABELLE 32 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{SAVEATACMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	49
TABELLE 33 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{SCANCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	50
TABELLE 34 „AUSGEWÄHLTE METRIKEN DER KLASSE T _{AREASCANCMD} “ (QUELLE: TOGETHER [®] , VERSION 6.0).....	51

VII.4 Abbildungen

ABBILDUNG 1 „UML-KLASSENDIAGRAMM: SUBSYSTEM ABLAUFSTEUERUNG“ (QUELLE: TOGETHER [®] , VERSION 6.0).....	10
ABBILDUNG 2 „METHODENAUF RUF E WÄHREND DER EINZELKOMMANDO- BZW. MAKROVERARBEITUNG“ (QUELLE: SELBST).....	16
ABBILDUNG 3 „SCHEMATISCHE DARSTELLUNG DER ABLÄUFE BEI T _{GOTOINTENSITYCMD} “ (QUELLE: SELBST).....	34
ABBILDUNG 4 „SCHEMATISCHE DARSTELLUNG DER ABLÄUFE BEI T _{GOTOPEAKCMD} “ (QUELLE: SELBST).....	38