DESIGNBESCHREIBUNG ZUM REENGINEERING DER TOPOGRAPHIE

Dokument zur Diplomarbeit
- Designphase -

Autoren	Thomas Kullmann, Günther Reinecker
Dokumentversion	1.4
Zustand	abgeschlossen
letzte Bearbeitung	25.01.04

Inhaltsverzeichnis

I Überblick

Die ursprüngliche Topografie zeigte nach eingehender Bewertung einige Mängel. Diese reichen von einer schlechten Ergonomie der Oberfläche bis zu konkreten Designfehlern. Da es sich gezeigt hat, dass die Topografie einen wichtigen Anteil am Arbeitsprozess hat, erfolgt nun ein Re-Engineering.

Der Kern dieser Überarbeitung bildet die klare Trennung von Funktionalität und Oberfläche. Dies schien besonders wichtig, da bereits eine Klasse für die Funktionalität existierte, die aber nur die Sicherung einiger Datenelemente übernahm. Die eigentliche Steuerung war aber direkt in den Oberflächenklassen implementiert.

Das Dokument beinhaltet eine vollständige Auflistung der verwendeten Klassen, Strukturen, Attribute und Methoden, die bei der Implementation der Funktionalität oder dem Oberflächenentwurf Verwendung finden. Diese wurde durch einheitliche Layoutkonventionen stark formalisiert (siehe **[6]**). Die einzelnen Elemente sind fett hervorgehoben und werden jeweils kurz erläutert, wobei auch auf Zusammenhänge untereinander hingewiesen wird. Bei grundlegenden Verständnisproblemen sollte das zugehörige Dokument zum Messablauf (siehe **[1]**) mit hinzugezogen werden.

Die wichtigste Eigenschaft ist die klare Trennung von Funktionalität und Oberfläche (siehe **Abbildung 1 Klassendiagramm der neuen getrennten Topographie**). Das entstandene Design soll einen problemlosen Austausch der Oberfläche ermöglichen, ohne die Implementation der Funktionalität zu verändern. Zur Anbindung wird innerhalb der Dialogfensterklassen dynamisch genau ein Objekt (Singleton-Mechanismus) der Funktionalität erzeugt, über das die gesamte Verwaltung und Steuerung der Antriebe abrufbar ist.

Abbildung 1 Klassendiagramm der neuen getrennten Topographie

II Funktionalität

II.1 Strukturen

‣ **`enum EFormat { eDF, eSF }`** **GLOBAL NEU**

ist der Aufzählungstyp zur Differenzierung der Anzahl der Nachkommastellen von Antriebsparametern. Er wird nur von der Methode `LPCSTR GetUnitFormat( EFormat )` verwendet (zum Verständnis der Antriebe siehe [2]).

Element	Nachkommastellen
eDF	Zahlenformat wie im Antrieb
eSF	eine Nachkommastellen mehr

Tabelle 1 Beschreibung der Elemente von EFormat

II.2 Klasse TTopography

Deklaration : TP_FUNK.H
Implementation: TP_FUNK.CPP

II.2.1 Attribute

‣ **`static TTopography *m_Instance`** **PRIVATE NEU**

ist die einzige Instanz der Klasse `TTopography und` wird, wenn noch nicht vorhanden, durch die Methode `GetInstance(void)` dynamisch erzeugt.

‣ **`TDetector *m_lnkDetector`** **PRIVATE**

beinhaltet die Referenz auf den aktuellen Detektor.

‣ **`Tdetector *m_lnkMonitor`** **PRIVATE**

soll die Referenz auf einen Monitor-Detektor beinhalten, falls ein solcher irgendwann existiert. **Dies entspricht im Moment dem aktuellen Detektor `m_lnkDetector`.**

‣ **`TMacroTag *m_lnkMWorkPoint`** **PRIVATE**

beinhaltet das Makro zum Anfahren des Arbeitspunkts (zur Makroverarbeitung siehe [3]).

‣ **`BOOL m_bSmallAngleSide`** **PRIVATE**

zeigt an ob nach links oder rechts gefahren werden soll, d.h. Bewegung entweder in Richtung Minimal- oder Maximalposition.

‣ **`WORD m_nNumberCycle`** **PRIVATE**

ist die Anzahl der Mess-Zyklen. Diese ist bei Einfachbelichtung stets `1`. Nur bei Mehrfachbelichtung kann die Anzahl größer als `1` sein.

‣ **`float m_fWorkPoint`** **PRIVATE**

beinhaltet die Antriebsposition für den Arbeitspunkt in Winkelwerten - Einheit und Genauigkeit sind antriebsabhängig.

float m_fControlRange **PRIVATE**
beinhaltet den Regel-Bereich nach der Intensität und wird hier als Parameter für das Kommando `ControlFlank` zur Nachregelung verwendet.

float m_fControlStep **PRIVATE**
ist die Schrittweite in Winkelwerten zur Nachregelung und wird beim Anfahren im ausgewählten Antrieb gesetzt.

float m_fMoveStep **PRIVATE**
ist die Schrittweite in Winkelwerten zum Anfahren des Arbeitspunktes und wird beim Antrieb gesetzt bevor das Makro `MoveToPoint` ausgeführt wird (zur Makroverarbeitung siehe [3]).

float m_fMaxAngleEscape **PRIVATE**
ist die maximal erlaubte Abweichung vom Arbeitspunkt in Winkelwerten.

LONG m_lMeasurementTime **PRIVATE**
zeigt die wirkliche Dauer einer Messung in Sekunden. Eine Messung ist hierbei ein Zyklus.

int m_Motor **PRIVATE**
ist der Index des ausgewählten Antriebs. Alle Bewegungen werden nur mit diesem ausgeführt. Die Initialisierung erfolgt mittels `DoInitMotor` (zum Verständnis der Antriebe siehe [2]).

float m_fMaxTime **PRIVATE**
ist die maximale Messzeit und wird an den ausgewählten Detektor übergeben.

DWORD m_dwMaxCounts **PRIVATE**
ist die maximale Anzahl an Impulsen und wird an den ausgewählten Detektor übergeben.

BOOL m_bMonitorUsed **PRIVATE**
zeigt die Verwendung eines Monitor-Detektors. Dieser kann als zusätzlicher Detektor zur Überwachung der Messung angeschlossen werden.

BOOL m_bMultipleShot **PRIVATE**
zeigt ob eine Mehrfach-Belichtungen durchgeführt wird - Mehrfachbelichtung ↔ `TRUE`.

double m_dStartAngle **PRIVATE**
ist die Startposition in Winkelwerten relativ zum *Peak*.

float m_fAngleBetweenShots **PRIVATE**
ist der Abstand in Winkelwerten zwischen den einzelnen Mess-Zyklen bei einer Mehrfachbelichtung. Am Anfang eines neuen Zyklus wird mit dem Kommando `MoveToPoint` und `m_fAngleBetweenShots` als Kommando-Parameter die neue Antriebsposition angefahren. (zur Makroverarbeitung siehe [3]).

BOOL m_bControlActive **PRIVATE**
zeigt an, ob eine Messung läuft.

BOOL m_bSetupOk **PRIVATE**
gibt an, ob alle Einstellungen korrekt sind, d.h. hauptsächlich ob ein Antrieb und ein Detektor ausgewählt sind.

BOOL m_bStartPointOk **PRIVATE**
zeigt an, ob vor dem Beginn der Messung der Arbeitspunkt korrekt angefahren wurde.

WORD m_RestShots **PRIVATE**
ist die Anzahl der Mess-Zyklen, die bei einer Mehrfachbelichtung noch durchzuführen sind. Zu Beginn einer Mehrfachbelichtung gilt demnach: `m_RestShots == m_nNumberCycle.`

BOOL m_bTimeFinish **PRIVATE**
zeigt an, ob die Messzeit abgelaufen ist.

BOOL m_bAdditionalTime **PRIVATE**
gibt an, dass zusätzliche Messzeit vorhanden ist. Diese entsteht, wenn die Messung länger als `m_fMaxTime` dauert.

BOOL m_bExceptionOccured **PRIVATE**
zeigt an, dass während einer Messung ein Fehler aufgetreten ist.

BOOL m_bTimeRunning **PRIVATE**
zeigt an, ob ein Timer gestartet wurde, der die Messzeit überwacht.

DWORD m_dwCurrentTime **PRIVATE**
beinhaltet die aktuelle Messzeit in Millisekunden.

DWORD m_dwStartTime **PRIVATE**
beinhaltet den Zeitpunkt beim Start einer Messung in Millisekunden.

II.2.2 Methoden

TTopography (void) **PRIVATE**
ist der Standard-Konstruktor - außerhalb der Klasse nicht anwendbar (Singleton-Mechanismus).

TTopography (const TTopography&) **PRIVATE NEU**
ist der Kopier-Konstruktor - außerhalb der Klasse nicht anwendbar (Singleton-Mechanismus).

TTopography& operator = (const TTopography&) **PRIVATE NEU**
der Zuweisungsoperator - außerhalb der Klasse nicht anwendbar (Singleton-Mechanismus).

static TTopography *GetInstance (void) **PUBLIC NEU**
gibt eine Referenz auf die einzige Instanz der Klasse `TTopography` zurück. Existiert diese noch nicht wird sie dynamisch erzeugt (Singleton-Mechanismus).

TDetector *GetDetector (void) **PUBLIC NEU**
gibt `m_lnkDetector` zurück.

TDetector *GetActDetector (void) PUBLIC NEU
gibt eine Referenz auf den aktuellen Detektor zurück (zur Detektornutzung siehe [5]).

TDetector *GetMonitor (void) PUBLIC NEU
gibt `m_lnkMonitor` zurück.

TMacroTag *GetMWorkPoint (void) PUBLIC NEU
gibt `m_lnkWorkPoint` zurück.

int GetMotor (void) PUBLIC NEU
gibt `m_Motor` zurück.

LPCSTR GetMotorName (int) PUBLIC NEU
gibt die Bezeichnung eines Antriebs zurück. Der Parameter ist der Index in der Antriebsliste `TMList`. Wenn der Index falsch war oder das Motorobjekt nicht existiert. wird ein Leer-String zurückgegeben (zur Antriebssteuerung siehe [2]).

LPCSTR GetMotorUnit (void) PUBLIC NEU
gibt die Einheit des aktuellen Antriebs zurück.

LPCSTR GetUnitFormat (Eformat) PUBLIC NEU
gibt das Zahlenformat für die Parameter des aktuellen Motors zurück (siehe dazu `struct EFormat`) und wird zur Formatierung der Eingabeparameter bei der Makroverarbeitung benutzt.

int GetActMotor (void) PUBLIC NEU
gibt den Index des aktuellen Antriebs zurück. Sollte das dazugehörige Motorobjekt nicht existieren, wird `-1` zurückgegeben (zur Antriebssteuerung siehe [2]).

int GetMotorCount (void) PUBLIC NEU
gibt `m_MotorCount` zurück.

double GetStartAngle (void) PUBLIC NEU
gibt `m_dStartAngle` zurück.

double GetAngle (void) PUBLIC NEU
gibt die Position des aktuellen Antriebs in Winkelwerten zurück. Ist dieser nicht verfügbar, wird `0.0` zurückgegeben (zur Antriebssteuerung siehe [2]).

UINT GetMotorDigits (EFormat) PUBLIC NEU
gibt die Anzahl der Nachkommastellen für die Parameter des aktuellen Antriebs zurück (zur Antriebssteuerung siehe [2]).

LONG GetMeasurementTime (void) PUBLIC NEU
gibt `m_lMeasurementTime` zurück.

DWORD GetCurrentTime (void) PUBLIC NEU
gibt `m_dwCurrentTime` zurück.

DWORD GetStartTime (void) **PUBLIC NEU**
gibt `m_dwStartTime` zurück.

float GetMaxTime (void) **PUBLIC NEU**
gibt `m_fMaxTime` zurück.

DWORD GetMaxCounts (void) **PUBLIC NEU**
gibt `m_dwMaxCounts` zurück.

WORD GetNumberCycle (void) **PUBLIC NEU**
gibt `m_nNumberCycle` zurück.

WORD GetRestShots (void) **PUBLIC NEU**
gibt `m_nRestShots` zurück.

float GetControlStep (void) **PUBLIC NEU**
gibt `m_fControlStep` zurück.

float GetControlRange (void) **PUBLIC NEU**
gibt `m_fControlRange` zurück.

float GetMoveStep (void) **PUBLIC NEU**
gibt `m_fMoveStep` zurück.

float GetMaxAngleEscape (void) **PUBLIC NEU**
gibt `m_fMaxAngleEscape` zurück.

float GetWorkPoint (void) **PUBLIC NEU**
gibt `m_fWorkPoint` zurück.

float GetAngleBetweenShots (void) **PUBLIC NEU**
gibt `m_fAngleBetweenShots` zurück.

BOOL SetMotor (TAxisType) **PUBLIC NEU**
setzt den aktuellen, lokal verwendeten Motor in `m_Motor`. Der Parameter ist die standardisierte, texturelle Antriebsbezeichnung. Ist das Setzen nicht möglich, bleibt der bisherige Antrieb bestehen - bei Erfolg `SetMotor` ↔ `TRUE` (zur Antriebssteuerung siehe [2]).

BOOL SetMotor (int) **PUBLIC NEU**
setzt den aktuellen, lokal verwendeten Motor in `m_Motor`. Der Parameter ist der Index des Antriebs in `TMList`. Ist das Setzen nicht möglich, bleibt der bisherige Antrieb bestehen - bei Erfolg `SetMotor` ↔ `TRUE` (zum Verständnis der Antriebe siehe [2]).

void SetDetector (TDetector*) **PUBLIC NEU**
setzt `m_lnkDetector`.

▸ void SetMonitor (TDetector*) PUBLIC NEU
setzt `m_lnkMonitor`.

▸ BOOL SetMeasurementTime (LONG) PUBLIC NEU
setzt `m_lMeasurementTime`. Nur wenn der Wert des Parameters im Intervall [`1`, `9999999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben.

▸ void SetStartTime (DWORD) PUBLIC NEU
setzt `m_dwStartTime`.

▸ void SetCurrentTime (DWORD) PUBLIC NEU
setzt `m_dwCurrentTime`.

▸ BOOL SetMaxTime (float) PUBLIC NEU
setzt `m_fMaxTime`. Nur wenn der Wert des Parameters im Intervall [`0.1`, `99999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Kleinere `float`-Werte würden bei einer erneuten Darstellung fehlerhaft angezeigt werden.

▸ BOOL SetNumberCycle (LONG) PUBLIC NEU
setzt `m_nNumberCycle`. Nur wenn der Wert des Parameters im Intervall [`1`, `65535`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben.

▸ BOOL SetMaxCounts (LONG) PUBLIC NEU
setzt `m_dwMaxCounts`. Nur wenn der Wert des Parameters im Intervall [`0`, `999999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben.

▸ WORD SetRestShots (WORD) PUBLIC NEU
setzt `m_nRestShots`.

▸ BOOL SetControlRange (float) PUBLIC NEU
setzt `m_fControlRange`. Nur wenn der Wert des Parameters im Intervall [0.01, 9999.0] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Kleinere `float`-Werte würden bei einer erneuten Darstellung fehlerhaft angezeigt werden.

▸ BOOL SetAngleBetweenShots (float) PUBLIC NEU
setzt `m_fAngleBetweenShots`. Nur wenn der Wert des Parameters in den Intervallen [`-9999`, `-0.1`] oder [`0.1`, `99999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Die `float`-Werte außerhalb dieser Intervalle würden bei einer erneuten Darstellung fehlerhaft angezeigt werden.

▸ BOOL SetStartAngle (double) PUBLIC NEU
setzt `m_dStartAngle`; Nur wenn der Wert des Parameters im Intervall [`-9999`, `99999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben.

BOOL SetMoveStep (float) PUBLIC NEU
setzt `m_fMoveStep`. Nur wenn der Wert des Parameters im Intervall [`<min>`, `99999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Die Genauigkeit von `<min>` hängt vom gewählten Antrieb ab. Sie ist dabei stets um eine Nachkommastelle höher als im Antrieb (zum Verständnis der Antriebe siehe [2]).

BOOL SetControlStep (float) PUBLIC NEU
setzt m_fControlStep. Nur wenn der Wert des Parameters im Intervall [`<min>`, `9999999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Die Genauigkeit von `<min>` hängt vom gewählten Antrieb ab. Sie ist dabei aber stets um eine Nachkommastelle höher als im Antrieb (zum Verständnis der Antriebe siehe [2]).

BOOL SetAngleWidth (float) PUBLIC NEU
setzt die Schrittweite im aktuellen Motor. Ist dies nicht möglich, bleibt die bisherige bestehen - bei Erfolg `SetAngleWidth ↔ TRUE`.

BOOL SetMaxAngleEscape (float) PUBLIC NEU
setzt `m_fMaxAngleEscape`. Nur wenn der Wert des Parameters im Intervall [`0.1`, `9999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Kleinere `float`-Werte würden bei einer erneuten Darstellung fehlerhaft angezeigt werden.

BOOL SetWorkPoint (float) PUBLIC NEU
setzt `m_fWorkPoint`. Nur wenn der Wert des Parameters in den Intervallen [`-9999`, `-0.1`] oder [`0.1`, `99999`] liegt, wird das Attribut geschrieben und `TRUE` zurückgegeben. Die `float`-Werte außerhalb der Intervalle würden bei einer erneuten Darstellung fehlerhaft angezeigt werden.

void SetTimeFinish (BOOL) PUBLIC NEU
setzt `m_bTimeFinish`.

void SetTimeRunning (BOOL) PUBLIC NEU
setzt `m_bTimeRunning`.

void SetAdditionalTime (BOOL) PUBLIC NEU
setzt `m_bAdditionalTime`.

void SetStartPointOk (BOOL) PUBLIC NEU
setzt `m_bStartPointOk`.

void SetExceptionOccured (BOOL) PUBLIC NEU
setzt `m_bExceptionOccured`.

void SetControlActive (BOOL) PUBLIC NEU
setzt `m_bControlActive`.

void SetMonitorUsed (BOOL) PUBLIC NEU
setzt `m_bMonitorUsed`.

▸ void SetMultipleShot (BOOL) **PUBLIC NEU**
setzt `m_bMultipleShot`.

▸ void SetSmallAngleSide (BOOL) **PUBLIC NEU**
setzt `m_bSmallAngleSide`.

▸ void SetSetupOk (BOOL) **PUBLIC NEU**
setzt `m_bSetupOk`.

▸ BOOL HasMotor (void) **PUBLIC NEU**
zeigt an ob der lokale Motor gesetzt ist - `TRUE ↔ m_Motor > -1`.

▸ BOOL HasAdditionalTime (void) **PUBLIC NEU**
gibt `m_bAdditionalTime` zurück.

▸ BOOL IsMultipleShot (void) **PUBLIC NEU**
gibt `m_bMultipleShot` zurück.

▸ BOOL IsControlActive (void) **PUBLIC NEU**
gibt `m_bControlActive` zurück.

▸ BOOL IsSmallAngleSide (void) **PUBLIC NEU**
gibt `m_bSmallAngleSide` zurück.

▸ BOOL IsSetupOk (void) **PUBLIC NEU**
gibt `m_bSetupOk` zurück.

▸ BOOL IsStartPointOk (void) **PUBLIC NEU**
gibt `m_bStartPointOk` zurück.

▸ BOOL IsStarted (void) **PUBLIC NEU**
gibt `m_bStarted` zurück.

▸ BOOL IsMonitorUsed (void) **PUBLIC NEU**
gibt `m_bMonitorUsed` zurück.

▸ BOOL IsTimeRunning (void) **PUBLIC NEU**
gibt `m_bTimeRunning` zurück.

▸ BOOL IsExeptionOccured (void) **PUBLIC NEU**
gibt `m_bExceptionOccured` zurück.

▸ void SetStrngStartPoint (double) **PUBLIC NEU**
setzt `Steering.dStartPoint`.

▸ double GetStrngStartPoint (void) PUBLIC NEU
gibt `Steering.dStartPoint` zurück.

▸ double GetStrngDistance (void) PUBLIC NEU
gibt `Steering.GetDistance()` zurück.

▸ BOOL DetectorRequest (void) PUBLIC NEU
gibt `Steering.DetectorRequest()` zurück.

▸ float GetStrngPeakIntensity (void) PUBLIC NEU
gibt `Steering.fPeakIntensity` zurück.

▸ float GetStrngStartIntensity (void) PUBLIC NEU
gibt `Steering.fStartIntensity` zurück.

▸ float GetStrngIntensity (void) PUBLIC NEU
gibt `Steering.GetIntensity()` zurück.

▸ float GetStrngHwb (void) PUBLIC NEU
gibt `Steering.GetHwb()` zurück.

▸ BOOL DoInitDetectors (TDetector*) PUBLIC NEU
initialisiert den lokalen Detektor und Monitor mit dem Wert des Parameters, wenn diese noch nicht initialisiert wurden - Rückgabe von `TRUE` ↔ <lokale Detektor> `!= NULL` (zur Detektornutzung siehe [5]).

▸ BOOL DoInitMotor (TAxisType, BOOL) PUBLIC NEU
initialisiert den lokalen Antrieb mit dem Wert des ersten Parameters, wenn dieser noch nicht initialisiert wurde. Schlägt dies fehl, wird versucht den lokalen auf den aktuellen Antrieb zu setzen. Der zweite Parameter entscheidet, ob die Antriebsdaten in der `Settings`-Struktur gesichert werden sollen - Rückgabe von `TRUE` ↔ lokaler Antrieb `!= NULL` (zum Verständnis der Antriebe siehe [2]).

▸ void DoInitWorkPoint (void) PUBLIC NEU
ist die Initialisierung der Ablaufsteuerung, d.h. des `Steering`-Objekts zum Anfahren des Arbeitspunktes (zur Makroverarbeitung siehe [3]).

▸ BOOL DoStartMeasure (TCmdId, int, int, LPSTR, HWND) PUBLIC NEU
ist zur Durchführung einer Messung mittels Kommandoverarbeitung. 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 Funktion `Steering.StartCmdExecution`. Der letzte Parameter wird `hControlWnd` zugewiesen (zur Makroverarbeitung siehe [3]).

‣ **BOOL DoStartMeasure (TMacroTag*, HWND) PUBLIC NEU**
ist zur Durchführung einer Messung mittels Makroverarbeitung. Der erste Parameter ist die Makrobezeichnung und der zweite Parameter wird `hHostWindow` zugewiesen. Die Verarbeitung geschieht mittels `Steering.StartMacroExecution.`

‣ **BOOL DoStopMeasure (void) PUBLIC NEU**
unterbricht eine laufende Makroausführung mittels `Steering.ToggleInterrupt()` – eine Fortsetzung der Messung ist möglich.

‣ **void DoSetDetectorParams (float, DWORD, BOOL) PUBLIC NEU**
setzt die Parameter für die Belichtung. Der erste Parameter ist die maximale Messzeit, der Zweite die maximalen Wiederholungen. War während des Aufrufs der Funktion eine Messung aktiv oder ist der dritte Parameter `== TRUE`, dann wird diese zuerst gestoppt und nach dem Setzen fortgesetzt (zur Detektornutzung siehe [5]).

‣ **void DoResetDetectorParams (void) PUBLIC NEU**
löscht die im Detektor gespeicherten Daten zur Topografie. War während des Aufrufs der Funktion eine Messung aktiv, wird diese gestoppt und nach dem Löschen nicht fortgesetzt (zur Detektornutzung siehe [5]).

‣ **BOOL DoSaveMonitorSignal (void) PUBLIC NEU**
normiert und speichert das Messsignal am Monitor-Detektor (zur Detektornutzung siehe [5])

‣ **BOOL DoSaveMotorSettings (int) PUBLIC NEU**
trägt die Werte eines Antriebs in die allgemeine `Settings`-Struktur ein. Der Parameter ist der Index des zu speichernden Antriebs. Ist der Index falsch, werden die Daten des Aktuellen gesichert (siehe dazu [2] `TMotor::PopSettings( aParameter )`).

‣ **BOOL DoLoadMotorSettings (int, TMParameter) PUBLIC NEU**
lädt die Werte aus der allgemeinen `Settings`-Struktur. Der erste Parameter ist der Index des zu speichernden Antriebs, der Zweite ist die lokale Ziel-Struktur. Ist der Index falsch, werden die Daten des Aktuellen geladen (siehe dazu [2] `TMotor::PushSettings()`).

II.2.3 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚**L**ines **O**f **C**ode' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	419
	‚**LOC** of **I**mplementation'*	LOCI	256
	‚**LOC** of **D**eclaration'*	LOCD	161
	‚**N**umber **O**f **A**ttributes'	NOA (0, 30)	**29**
	‚**N**umber **O**f **O**perations'	NOO (0, 50)	**86**
	‚**N**umber **O**f **M**embers' Attribute + Methoden	NOM = NOA + NOO	**115**
	‚**N**umber **O**f **C**onstructors'	NOCON (0, 5)	2
	‚**N**umber **O**f **O**verridden Methods'	NOOM (0, 10)	0
	‚**P**ercentage of **Priv**ate **M**embers'	PPrivM	28
	‚**P**ercentage of **Prot**ected **M**embers'	PProtM (0, 10)	0
	‚**P**ercentage of **Pub**lic **M**embers'	PPubM	73
	‚**W**eighted **M**ethods **P**er **C**lass'	WMPC1 (0, 30)	
Attribute			
	‚**A**ttribute **C**omplexity'	AC	206
Methoden			
	‚**M**aximum **N**umber **O**f **P**arameters'	MNOP (0, 4)	**5**
	‚**C**yclomatic **C**omplexity'	CC	
Kommentare			
	‚**N**umber **O**f **C**omments'*	NOC	209
	‚**T**rue **C**omment **R**atio'	TCR (5, 400)	42

Tabelle 2 Ausgewählte Metriken der Klasse `TTopography` (Quelle: Together ‚Version 5.5)

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

Anmerkungen zur Fehleranfälligkeit und -toleranz der Klasse `TTopography` findet man - wenn nötig - beim jeweiligen Member rot hervorgehoben.

II.3 Attribute

▶ **`extern TSteering Steering`** **GLOBAL**

Dieses Objekt kann nur Makroverarbeitung benutzt werden (zur Makroverarbeitung siehe [3]).

III Oberfläche

III.1 Klasse TTopographyExecDlg

Deklaration : TP_GUI.H

Implementation: TP_GUI.CPP

Die Klasse `TTopographyExecDlg` ist von der Basisklasse für modale Dialogfenster `TModalDlg` abgeleitet und erbt das Interface `ITimer` ein.

III.1.1 Attribute

▸ **`TTopography *m_lnkTopography`** **PRIVATE NEU**

ist der Zeiger auf die Funktionalität. Das referenzierte Objekt enthält die gesamte Funktionalität zur Steuerung der Topografie. Es wird dynamisch im Konstruktor des Dialogfensters erzeugt oder, wenn die Funktionalität schon existiert, mit dieser verknüpft.

TTimer *m_lnkTimerMeasuring **PRIVATE NEU**
ist der dynamisch im Konstruktor des Dialogfensters erzeugte Timer zur Überwachung der Gesamtzeit einer Messung.

TTimer *m_lnkTimerMultiExpose **PRIVATE NEU**
ist der dynamisch im Konstruktor des Dialogfensters erzeugte Timer zur Überwachung der Dauer einer Mehrfachbelichtung.

III.1.2 Methoden

TTopographyExecDlg (void) **PUBLIC**
ist der Standard-Konstruktor. Zur Anzeige des dazugehörigen Dialogfensters wird der Konstruktor der Basisklasse `TModalDlg` gerufen (siehe [4], für den zu übergebenden Parameter siehe **III.1.3**). Hier wird die Funktionalität verknüpft, die Timer erzeugt und Antrieb sowie Detektor initialisiert.

virtual BOOL Dlg_OnInit (HWND, HWND, LPARAM) **PUBLIC**
ist zur Initialisierung bevor das Dialogfenster angezeigt wird. Es werden einige Parameter der Funktionalität initialisiert und, wenn nötig, ein neues Zählerfenster erzeugt. Es folgen die Aufrufe weiterer Initialisierungs-Funktionen (`Dlg_OnInitialize()` und `Dlg_OnParamSet()`). Dies ist eine Methode der Basisklasse `TBasicWindow` (siehe [4]) - Rückgabewert ist immer `TRUE`.

virtual void Dlg_OnCommand (HWND, int, HWND, UINT) **PUBLIC**
ist zur Verarbeitung alle Steuerelement-Ereignisse des Dialogfensters und zum Aufruf der entsprechenden Behandlungsroutinen. Dies ist eine Methode der Basisklasse `TModalDlg` (siehe [4]).

virtual void OnTimer (const TBasicTimer*) **PUBLIC**
dient zur Ereignisbehandlung für die beiden verwendeten Timer. Über den Parameter können die Timer separat behandelt werden. Wenn `m_lnkTimerMeasuring` zuschlägt, erfolgt die Ausgabe eines regelmäßigen Pieptons. Bei `m_lnkTimerMultiExpose` erfolgt ein erneuter Aufruf der Methode zur Mehrfachbelichtung.

virtual BOOL LeaveDialog (void) **PUBLIC**
wird beim Verlassen des Dialogfensters ausgeführt. Dies hat die Freigabe der Zeiger auf die Timer und die Funktionalität sowie das Zurücksetzen der Parameter im aktuellen Antrieb und Detektor zur Folge. Dies ist eine Methode der Basisklasse `TBasicDialog` (siehe [4]).

void Dlg_OnMultipleExpose (void) **PUBLIC NEU**
dient zur Behandlung des Timeouts bei Mehrfachbelichtung. Sind noch Messzyklen übrig, werden Antrieb und Detektor neu initialisiert und der nächste Arbeitspunkt angefahren. Ansonsten erfolgt der Aufruf von `Dlg_OnSwitchControl`.

void Dlg_OnSwitchControl (void) **PUBLIC NEU**
dient der Behandlung der Nachricht `cm_SwitchControl`. Es werden drei Ereignisse unterschieden:

- eine Messung ist beendet → Herstellung des Ausgangszustandes
- Einfachbelichtung wurde gestartet → Ausführung des Kommandos `ControlFlank`
- Mehrfachbelichtung wurde gestartet → Restzyklen setzen und Timer starten

Void Dlg_OnInitialize (void) PUBLIC NEU
initialisiert den Detektor mit einem Zeitintervall, einer maximalen Impulsrate und dem Messfehlerwert (0.02). Es wird eine Warnung ausgegeben, falls Antrieb oder Detektor vor dem Aufruf von 'Topographie Ausführen' nicht gesetzt sind.

Void Dlg_OnGotoWorkPoint (void) PUBLIC NEU
dient zur Behandlung der Nachricht `cm_GotoWorkPoint`. Bei Mehrfachbelichtung wird der um `dStartAngle` vom aktuellen Stand entfernte Punkt mittels Kommandoverarbeitung (`MoveToPoint`) angefahren. Bei Einfachbelichtung wird der Arbeitspunkt mit dem Makro „`SetupTopography`" angefahren (zur Makroverarbeitung siehe [3]).

void Dlg_OnSetupPosition (void) PUBLIC NEU
ist zur Behandlung der Nachricht `cm_SetupPosition`. Dieses Ereignis tritt ein, wenn der Arbeitspunkt korrekt angefahren wurde. Es wird somit angezeigt, dass jetzt eine Messung gestartet werden kann. Bei einer Mehrfachbelichtung wird der Startpunkt im `Steering`-Objekt aktualisiert.

void Dlg_OnSteeringReady (void) PUBLIC NEU
ist zur Behandlung der Nachricht `cm_SteeringReady`. Diese wird vom `Steering`-Objekt ausgelöst. War die `Steering`-Aktion bezüglich eines Umsetzungsschritts bei einer Mehrfachbelichtung erfolgreich, wird der Timer für die nächste Runde neu initialisiert. Ansonsten erfolgt ein Aufruf von `Dlg_OnSetupPosition` (zur Detektornutzung siehe [5]).

void Dlg_OnCounterSet (void) PUBLIC NEU
ist zur Behandlung der Nachricht `cm_CounterSet` und dient bei einer Einfachbelichtung zur Aktualisierung der Anzeige des Zählerfensters (zur Detektornutzung siehe [5]).

void Dlg_OnExposureTime (void) PUBLIC NEU
Dieses Ereignis tritt ein, wenn eine neue maximale Messzeit im Eingabefeld ‚Messzeit' eingegeben wurde. Dieser Wert wird an den Detektor übergeben.

void Dlg_OnTopographyParam (void) PUBLIC NEU
dient zur Behandlung der Nachricht `cm_TopographyParam`. Es wird das Dialogfenster 'Topographie Einstellungen' modal geöffnet. Nach dem Schließen des Fensters wird der Wert im Feld ‚Messzeit' aktualisiert.

void Dlg_OnParamSet (void) PUBLIC NEU
dient zur Behandlung der Nachricht `cm_ParamSet`. Bei aktiver Messung wird die abgelaufene Zeit neu berechnet und, wenn vorhanden, die zusätzliche Messzeit sowie die Drift des Antriebs ausgegeben.

void Dlg_OnInquireException (void) PUBLIC NEU
ist zur Behandlung der Nachricht `cm_InquireException`. Dabei werden folgende Fehlerfälle unterschieden und behandelt:

- Intensität unter Minimum
- Abweichung vom Arbeitspunkt ist zu groß
- Messzeit ist abgelaufen

III.1.3 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚**L**ines **O**f **C**ode' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	433
	‚**LOC** of **I**mplementation'*	LOCI	369
	‚**LOC** of **D**eclaration'*	LOCD	64
	‚**N**umber **O**f **A**ttributes'	NOA (0, 30)	3
	‚**N**umber **O**f **O**perations'	NOO (0, 50)	15
	‚**N**umber **O**f **M**embers' Attribute + Methoden	NOM = NOA + NOO	18
	‚**N**umber **O**f **C**onstructors'	NOCON (0, 5)	1
	‚**N**umber **O**f **O**verridden Methods'	NOOM (0, 10)	4
	‚**P**ercentage of **Priv**ate **M**embers'	PPrivM	74
	‚**P**ercentage of **Prot**ected **M**embers'	PProtM (0, 10)	0
	‚**P**ercentage of **Pub**lic **M**embers'	PPubM	26
	‚**W**eighted **M**ethods **P**er **C**lass'	WMPC1 (0, 30)	
Attribute			
	‚**A**ttribute **C**omplexity'	AC	27
Methoden			
	‚**M**aximum **N**umber **O**f **P**arameters'	MNOP (0, 4)	**4**
	‚**C**yclomatic **C**omplexity'	CC	
Kommentare			
	‚**N**umber **O**f **C**omments'*	NOC	179
	‚**T**rue **C**omment **R**atio'	TCR (5, 400)	36

Tabelle 3 Ausgewählte Metriken der Klasse `TTopographyExecDlg` (Quelle: Together ‚Version 5.5)

* Diese Metriken sind nicht Bestandteil von Together, sondern wurden manuell ermittelt.
Anmerkungen zur Fehleranfälligkeit und -toleranz der Klasse findet man beim jeweiligen Member.

III.1.4 Ressourcen

Ressourcen-ID	Element	Bezeichnung
`TopographyExecute`	Dialogfenster	‚Topographie'
`TopographyExecuteEng`	…	‚Topography'
`cm_GotoWorkPoint`	Schaltfläche	‚Startposition einstellen'
`cm_SwitchControl`	…	‚Regelung starten'
`cm_TopographyParam`	…	‚Einstellungen'
`IDCANCEL`	…	‚Abbrechen'
`id_RemainderCyclus`	Textfeld	‚Restzyklen'
`id_MeasurementTime`	…	‚aktuelle Messzeit'
`id_Angle`	…	‚Drift'
`id_RemainderTime`	…	‚zusätzliche Zeit'
`id_Text`	…	‚Statusinformationen
`id_ExposureTime`	Eingabefeld	‚Detektor-Zeit'

Tabelle 4 Ressourcen für das Dialogfenster `TopographyExecDlg(Eng)`

III.2 Klasse TTopographyAdjustDlg

Deklaration : TP_GUI.H
Implementation: TP_GUI.CPP

III.2.1 Attribute

- **`TTopography *m_lnkTopography`** **PRIVATE NEU**

ist der Zeiger auf die Funktionalität. Er wird dynamisch bei der Initialisierung des Dialogfensters erzeugt. Das Objekt enthält die gesamte Funktionalität zur Steuerung der Topographie.

- **`HWND m_hDetectorList`** **PRIVATE**

ist ein Handle auf das Kombinationsfeld mit der Detektorenliste.

- **`HWND m_hMotorList`** **PRIVATE**

ist ein Handle auf das Kombinationsfeld mit der Antriebsliste.

- **`BOOL m_bCtrlStatus`** **PRIVATE NEU**

zeigt an, ob das Dialogfester von `TTopographyDlg` aus gerufen wurde. War dies der Fall, sind ein Teil der Steuerelemente zu sperren und auszugrauen.

III.2.2 Methoden

- **`TTopographyAdjustDlg ( BOOL )`** **PUBLIC**

ist der Konstruktor. Wenn das Dialogfester von `TTopographyExecDlg` aus gerufen wurde, ist der Parameter = `TRUE`. Zur Anzeige des dazugehörigen Dialogfensters wird der Konstruktor der Basisklasse `TModalDlg` gerufen (siehe [4], für den dort zu übergebenden Parameter siehe **III.2.3**).

- **`virtual BOOL Dlg_OnInit ( HWND, HWND, LPARAM )`** **PUBLIC**

ist für Initialisierungen bevor das Dialogfenster angezeigt wird. Wird das Einstellungsfenster von `TTopographyExecDlg` aus aufgerufen, so ist nur für einige Werte die Eingabe möglich. Alle anderen Eingabefelder werden hier gesperrt und ausgegraut (siehe `m_bCtrlStatus`). Hier erfolgt der Aufbau der Antriebs- und Detektorenlisten in den Kombinationsfeldern. Dies ist eine Methode der Basisklasse `TBasicWindow` (siehe [4]) - Rückgabe immer `TRUE`.

- **`virtual void Dlg_OnCommand ( HWND, int, HWND, UINT )`** **PUBLIC**

dient zur Verarbeitung alle Steuerelement-Ereignisse des Dialogfensters und zum Aufruf der entsprechenden Behandlungsroutinen. Dies ist eine Methode der Basisklasse `TModalDlg` (siehe [4]).

- **`virtual BOOL CanClose ( void )`** **PUBLIC**

testet alle Eingabefelder, ob die Werte im Wertebereich liegen und übernimmt sie in die Funktionalität `TTopography`. Bei der Rückgabe von `TRUE` und dem Schließen des Dialogfensters waren alle Eingaben korrekt. Dies ist eine Methode der Basisklasse `TBasicDialog` (siehe [4]).

- **`virtual void LeaveDialog ( void )`** **PUBLIC**

wird beim Verlassen des Dialogfensters ausgeführt und gibt den Zeiger auf die Funktionalität frei. Es ist eine Methode der Basisklasse `TBasicDialog` (siehe [4])

▸ **void Dlg_OnParamSet (void) PUBLIC NEU**

ist zur Behandlung der Nachricht `cm_ParamSet`. Die Steuerelemente werden mit den entsprechenden Werten aus der Funktionalität gefüllt.

▸ **void Dlg_OnChooseMotor (UINT) PUBLIC NEU**

dient zur Behandlung der Nachricht `id_ChooseMotor`. Es erfolgt die Initialisierung des Antriebs, der in der Antriebsliste neu ausgewählt wurde (zur Antriebssteuerung siehe [2]).

▸ **void Dlg_OnChooseDetector (UINT) PUBLIC NEU**

dient der Behandlung der Nachricht `id_ChooseDetector`. Es erfolgt die Initialisierung des Detektors, der in der Detektorenliste neu ausgewählt wurde (zur Detektornutzung siehe [5]).

▸ **void Dlg_OnMultipleShot (void) PUBLIC NEU**

ist zur Behandlung der Nachricht `id_MultipleShot`. Bei Aktivierung des Kontrollfeldes ‚MultipleShot' werden die Eingabefelder im Bereich Mehrfachbelichtung freigegeben und die Felder ‚Arbeitspunkt', ‚Lage' und ‚Belichtungsregelung Bereich' gesperrt und ausgegraut. Bei Deaktivierung erfolgt die Umkehrung der Freigabe / Sperrung der Steuerelemente.

III.2.3 Bewertung

Metrik		Kennung (min,max)	Wert
Klasse			
	‚**L**ines **O**f **C**ode' inkl. Leer- und Kommentarzeilen	LOC (0, 1000)	186
	‚**LOC** of **I**mplementation'*	LOCI	146
	‚**LOC** of **D**eclaration'*	LOCD	40
	‚**N**umber **O**f **A**ttributes'	NOA (0, 30)	4
	‚**N**umber **O**f **O**perations'	NOO (0, 50)	8
	‚**N**umber **O**f **M**embers' Attribute + Methoden	NOM = NOA + NOO	12
	‚**N**umber **O**f **C**onstructors'	NOCON (0, 5)	1
	‚**N**umber **O**f **O**verridden Methods'	NOOM (0, 10)	4
	‚**P**ercentage of **Priv**ate **M**embers'	PPrivM	62
	‚**P**ercentage of **Prot**ected **M**embers'	PProtM (0, 10)	0
	‚**P**ercentage of **Pub**lic **M**embers'	PPubM	38
	‚**W**eighted **M**ethods **P**er **C**lass'	WMPC1 (0, 30)	
Attribute			
	‚**A**ttribute **C**omplexity'	AC	36
Methoden			
	‚**M**aximum **N**umber **O**f **P**arameters'	MNOP (0, 4)	**4**
	‚**C**yclomatic **C**omplexity'	CC	
Kommentare			
	‚**N**umber **O**f **C**omments'*	NOC	104
	‚**T**rue **C**omment **R**atio'	TCR (5, 400)	48

Tabelle 5 Ausgewählte Metriken der Klasse `TTopographyAdjustDlg` (Quelle: Together ‚Version 5.5)

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

Anmerkungen zur Fehleranfälligkeit und -toleranz der Klasse `TTopographyAdjustDlg` findet man (wenn nötig) beim jeweiligen Member, rot hervorgehoben.

III.2.4 Ressourcen

Ressourcen-ID	Element	Bezeichnung
`TopographyParam`	Dialogfenster	‚Topographie Einstellungen'
`TopographyParamEng`	...	‚Topography Adjustments'
`IDCANCEL`	Schaltfläche	‚Abbrechen'
`IDOK`	…	‚OK'
`id_ChooseMotor`	Kombinationsfeld	Antriebsauswahlliste
`id_ChooseDetector`	…	Detektorauswahlliste
`id_MaxAngleEscape`	Eingabefeld	‚Beschränkung auf'
`id_MaxTime`	…	‚max Zeit'
`id_MaxCounts`	…	‚max Counts'
`id_WorkPoint`	…	‚Lage'
`id_MoveStep`	…	‚Fahren mit'
`id_ControlStep`	…	‚Schritt'
`id_ControlRange`	…	‚Bereich'
`id_Shots`	…	‚Anzahl'
`id_StepWidth`	…	‚Schrittweite'
`id_From`	…	‚Startwert'
`id_MeasurementTime`	…	‚Belichtungszeit'
`id_MultipleShot`	Kontrollfeld	‚Mehrfach-Belichtung'

Tabelle 6 Ressourcen für das Dialogfenster `TopographyAdjustDlg(Eng)`

Anhang A – VERWANDTE DOKUMENTE

[1] „Topographie - Gesamtvorgang", Version 2.1 von U. Sacklowski

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

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

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

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

[6] „Layoutkonventionen und Steuerelemente", Version 1.3 von Thomas Kullmann und Günther Reinecker

Anhang B – TABELLEN

Anhang C – ABBILDUNGEN