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

Diplomarbeit

Automatisierung von Regressionstests eines Programms zur Halbleiter-Strukturanalyse

Betreuer: Prof. Dr. Klaus Bothe

Jens Hanisch Johann Letzel

25. November 2002

Erklärungen

Selbständigkeitserklärung

Hiermit versichern wir, dass wir die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Quellen erstellt haben.

Berlin, 01.10.2002 Johann Letzel Jens Hanisch

Einverständniserklärung

Hiermit erklären wir unser Einverständnis mit der öffentlichen Ausstellung unserer Diplomarbeit in der Bibliothek des Instituts für Informatik.

Berlin, 01.10.2002 Johann Letzel Jens Hanisch

Zusammenfassung

Die vorliegende Arbeit aktualisiert und erweitert die Ergebnisse unserer Studienarbeit aus dem Wintersemester 2001/2002. Es wurde ein Testsystem entwickelt, welches die notwendige Überprüfung aller grundsätzlichen Programmfunktionalitäten nach einer Quellcode-Aktualisierung unterstützt. Dieser Prozess, in der Softwaretechnik als *Regressionstest* bezeichnet, fällt sehr häufig an und wird weitestgehend durch die Abarbeitung von Skripten automatisiert. Die Steuerungssoftware als Blackbox-Testobjekt erhält ihre Eingabedaten über Konfigurationsdateien und die grafische Benutzerschnittstelle. Messageboxen, Dateien, Kurvengrafiken und Zustände von Dialogboxelementen müssen als Ergebnisse bzw. Ausgaben mit entsprechenden Erwartungswerten verglichen werden.

Die intensive Beschäftigung mit der Problematik führte stellenweise zur Ersetzung einiger Teile der Studienarbeit. So wurde zum Beispiel der Dateivergleich von seiner statischen Natur befreit und liest nun Toleranzen direkt aus den Solldateien. Auch das Quellcode-Design des Testsystems wurde vollkommen überarbeitet und objektorientiert reimplementiert, um häufig benutzte Funktionen in mehreren entwickelten Test-Komponenten wiederzuverwenden.

Den größten Teil dieser Arbeit bildet jedoch die Erweiterung des Testsystems zu einer gereiften Version, die nun neben einem einfachen Skriptinterpreter viele weitere nützliche Funktionen enthält. So können nun alle Dialog-Steuerelemente und Menüs aus den Ressourcen einer aktuellen Programmversion dem Entwickler von Testskripten zugänglich gemacht und die Ausführungsreihenfolge der Testskripte nach bestimmten Kriterien festgelegt werden.

Die Entwicklung von Skripten findet zudem auf einer höheren Sprachebene statt, so dass der Entwickler die interne Ressourcenverwaltung des Testobjektes nicht mehr kennen muss. Damit sind auch die Skripte unempfindlicher gegenüber Änderungen an den Ressourcen geworden, da sich das Testsystem

um die Zuordnung zwischen Steuerelementen und numerischen IDs kümmert. Außerdem können auf diese Weise nur Skripte erstellt werden, die tatsächlich vorhandene Dialogboxen, Menüs und Steuerelemente aus der aktuellen Programmversion aufrufen.

Eine weitere Funktion ermöglicht die automatische Generierung von Skriptdateien aus attributierten Klassifikationsbäumen. Mit geeigneten Kombinationstabellen lassen sich auf diese Weise Testsequenzen systematisch entwerfen, um Komponenten des XCTL-Systems eingehend zu testen.

An den definierten Testfällen für einen Regressionstest aus der Studienarbeit hat sich hingegen nicht sehr viel geändert. Neue Testfälle wurden für kürzlich fertiggestellte Funktionen entwickelt und ergänzen das Regressionstestpaket. Eine dynamische Überdeckungsanalyse hat ergeben, dass dabei nur wenige Funktionen des Steuerprogrammes übersehen wurden. Das Regressionstestpaket hat mit nunmehr 24 Grobtestfällen einen Zustand erreicht, in dem alle Programmfunktionen auf einer XCTL-Version vom 20.01.2002 mit den Möglichkeiten der Umgebungssimulation mindestens einmal aufgerufen werden.

Kapitelübersicht

Das erste Kapitel gibt einen groben Überblick der Problematik und motiviert den Einsatz eines Systems zur Unterstützung und Automatisierung von Regressionstests.

Das zweite Kapitel diskutiert verschiedene Teststrategien, um daraus für unser Problem eine geeignete herauszuarbeiten. Einsatz und Realisierung erfordern schließlich eine Untersuchung, wie zweckmäßig die Umgebungssimulation dafür ist. Ergebnis ist der Entwurf von Pflichtenheften für die zu entwickelnden Komponenten eines Testsystems.

Kapitel drei bildet den Hauptteil dieser Arbeit. Hier wird der Umgang mit dem entwickelten Testsystem detailliert beschrieben. Neben den definierten Testfällen findet man hier die Spezifikationen der Skriptkommandos, Dateivergleichskommandos und der Anweisungen attributierter Klassifikationsbäume. Die Benutzerleifäden für Testkomponenten werden aus Platzgründen als Anlagen zu dieser Arbeit beigelegt.

Das vierte Kapitel beschreibt das Design wesentlicher Kernkomponenten des Testsystems. Unter Berücksichtigung des Umfanges dieser Arbeit kann auf die Implementierung nicht genauer eingegangen werden. Alle Quellen liegen dokumentiert im CVS-Repository vor und sind bei Bedarf einzusehen.

Im fünften Kapitel wird das entwickelte Testsystem einschließlich der definierten Testfälle hinsichtlich der gestellten Aufgabe bewertet. Zur Überprüfung der Vollständigkeit der Testfälle wird dabei die Methode der Quellcode-Instrumentierung vorgestellt und angewendet.

Inhaltsverzeichnis

Kapitel 1

Einführung

Am Institut für Physik der HU-Berlin werden in der Arbeitsgruppe „Röntgenbeugung an Schichtsystemen“ unter Prof. Köhler kristalline Strukturen von Halbleitern untersucht. Das entwickelte Programm zur Ansteuerung der Messanlagen besitzt trotz Nutzung von C++ als Implementierungssprache eine ungünstige SW-Architektur, wodurch es nicht mehr wartbar und erweiterbar ist.

Aufgrund der Bitte um Unterstützung, wurde das Projekt „Software-Sanierung“ am Lehrstuhl für Softwaretechnik unter Leitung von Prof. Bothe gegründet, welches mit den Methoden des „Reengineering“ seit nunmehr vier Jahren die Steuerungssoftware aufbereitet.

„Reengineering“ umfasst neben dem „Reverse Engineering“ mit der Analyse der bestehenden Software auch ein intensives „Restructuring“ mit einhergehenden Veränderungen an der Programmstruktur. Dabei werden ständig Veränderungen und Verbesserungen an der Steuerungssoftware durch die Studenten vorgenommen und in die Quelltext-Versionskontrolle CVS eingespielt. Sogar völlig neue Programmfunktionalitäten, wie die kürzlich entstandene „Automatische Justage“, werden entwickelt und erweitern die ursprüngliche Software.

Innerhalb des Projektes wurden die Software-Entwicklungsphase „Analyse und Definition“ mit Verhaltensspezifikationen, UseCase-Diagrammen und Pflichtenheften, die „Design“-Phase mit UML-Diagrammen und Interface-Beschreibungen sowie die Phase der „Implementierung“ mit Code-Reviews und Metriken von den Mitarbeitern besonders intensiv bearbeitet.

Die wichtige Testphase spielte in der bisherigen Projektarbeit eher eine untergeordnete Rolle. Die Erfahrung zeigt, dass die Modifikation eines vorhandenen Programmes fehleranfälliger ist, als das Schreiben eines neuen Programmes. Ein ausführliches und begleitendes Testen ist für die Weiterentwicklung des Systems unumgänglich, da sonst Funktionalitäten aus früheren Versionen unbemerkt verloren gehen und nur sehr schwer wieder hergestellt werden könnten. In der hier zu Grunde liegenden Diplomarbeit möchten wir eine Lösung vorstellen, die sich dieser Problematik annimmt.

1.1 Begriffe

Für das weitere Verständnis dieser Arbeit sind zunächst einige Begriffe zu erläutern. Für einige Begriffe gibt der IEEE-Standard 829-1983 [13] allgemeingültige Definitionen vor. Beim Übersetzen ins Deutsche ist es uns dabei vielleicht nicht immer gelungen, einen sinngemäßen Ausdruck zu finden. Begriffe wie Testpaket, Testfall, Testschritt werden in der Literatur leider sehr unterschiedlich verwendet und nicht eindeutig definiert. Um Mißverständnissen vorzubeugen, werden Definitionen für diese Ausdrücke von uns selbst vorgenommen, um ihre Bedeutung in dieser Arbeit festzulegen. Ein Testschritt wird weiterhin in Aktionsschritte unterteilt, denen und ein oder mehrere Auswertungsschritte gegenüber stehen. Diese Festlegung ist Folge unserer Teststrategie, da die Auswertung der Testfälle schon während ihrer Durchführung und nicht am Ende stattfindet.

Definition Softwareobjekt (IEEE): *Quellcode, Objektcode, Jobsteuerungs-Code, Steuerungsdaten, oder mehrere dieser Objekte zusammen.*

Definition Testobjekt (IEEE): *Ein Softwareobjekt, welches getestet werden soll.*

In unserem Projekt wird als Testobjekt das untersuchte Anwendungsprogramm bezeichnet. Es umfasst neben dem ausführbaren Steuerprogramm auch alle zusätzlich benötigten Komponenten, wie Laufzeitbibliotheken (DLL's) und Umgebungsdateien (`TESTDEV.DAT`, `STANDARD.MAK`).

Definition Test (IEEE):
(1) Eine Menge von einem oder mehreren Testfällen, oder
(2) Eine Menge von einem oder mehreren Testverfahren, oder
(3) Eine Menge von einem oder mehreren Testfällen und Testverfahren.

Definition Testfall-Spezifikation (IEEE): *Dokument zur Spezifizierung von Eingaben, erwarteten Ausgaben und einer Menge von Bedingungen zur Ausführung des Testobjektes.*

Definition Testverfahren-Spezifikation (IEEE): *Dokument zur Spezifizierung einer Sequenz von Aktionen zur Durchführung eines Testes.*

Im IEEE-Standard wird zwischen Dokumenten zur Spezifikation von Testfällen und Testverfahren unterschieden. Die Durchführung einer manuellen Justage mit bestimmten Antriebs-Einstellungen erfordert zum Beispiel das Starten der Steuerungssoftware sowie das Aufrufen der notwendigen Dialogbox, um die Parameter einzutragen und die Justierung auszulösen. Ein Testfall beschreibt im allgemeinen nur die zu überprüfende Situation, also unter welchen Bedingungen und Eingaben welche Ausgaben erzielt werden. Die dazu notwendigen Schritte werden innerhalb der Testverfahren beschrieben. Wie im Abschnitt 3.3.2 begründet, findet diese Abgrenzung in unserem Projekt jedoch nicht statt. Unsere Testfälle bzw. Testfalldokumente beschreiben neben den Ein- und Ausgabendaten auch gleichzeitig alle notwendigen Schritte zur Durchführung. Der Begriff Testverfahren wird aus diesem Grund in dieser Arbeit nicht weiter verwendet. Die ausgearbeitete Testmethodik (Kapitel 2) erfordert für jeden Testfall die Durchführung einer Testsequenz.

Definition Testsequenz, Testschritt, Aktionsschritt: *Eine Testsequenz ist eine Folge von Testschritten, welche typischen Arbeitsschritten eines Benutzers am untersuchten Programm entsprechen. Jeder Testschritt kann dabei aus einer beliebigen Folge von Aktions- und Auswertungsschritten bestehen. Aktionsschritte führen, bis auf den letzten, zu keinen auswertbaren Reaktionen des Testobjektes und sind daher im allgemeinen in ihrer Ausführung vertauschbar. Im Unterschied zu den Aktionsschritten, sind die Testschritte voneinander abhängig, da jede Reaktion des Testobjektes zu neuen Ausgangsbedingungen für den nachfolgenden Testschritt führen kann.*

Definition Auswertungsschritt: *Schritte, die das Testsystem unternimmt, um eine auswertbare Reaktion des Testobjektes zu überprüfen, werden als Auswertungsschritte bezeichnet. Sie werden innerhalb eines Testschrittes gleich im Anschluss an die Aktionsschritte durchgeführt und sind in ihrer Reihenfolge vertauschbar.*

Die Unterscheidung zwischen Testschritten und Aktionsschritten ist vorteilhaft, weil innerhalb der Testsequenzen nur die Resultate und Ereignisse von Testschritten für eine Auswertung herangezogen werden müssen. Die so gewonnene Strukturierung einer Testsequenz läßt sich besonders gut verwalten und dokumentieren. Die Zuordnung von Aktionsschritten zu einem Testschritt ist davon abhängig, welche Reaktionen des Testobjektes als „auswertbar“ angesehen werden. Je nach den Möglichkeiten des Testsystems zur

Auswertung der Ergebnisse und in Abhängigkeit der Anforderungen des Entwicklers, können somit unterschiedlich viele Aktionsschritte einen Testschritt bilden.

Sollen zum Beispiel die Reaktionen von Dialogboxen auf bestimmte Eingaben in einer Testsequenz überprüft werden (Deaktivieren von Steuerelementen), so bilden alle Aktionsschritte bis zum Auftreten dieser GUI-Aktualisierung einen Testschritt. Legt man jedoch keinen Wert auf die Auswertung des Zustandes der Steuerelemente in einer Dialogbox, so bilden alle Aktionsschritte zum Ausfüllen der Parametermaske einen einzigen Testschritt. Aktionsschritte, die eine Veränderung der Benutzerschnittstelle wie das Deaktivieren von Steuerelementen hervorrufen und innerhalb eines Testschrittes liegen, sind nicht mehr in ihrer Reihenfolge vertauschbar, da alle nachfolgenden Aktionsschritte einer neuen Ausgangsituation unterliegen.

Wie sich im Laufe dieser Arbeit herausstellen wird, sind vor und nach der eigentlichen Durchführung einer Testsequenz noch zusätzliche Aktionen zum Umgang mit den Konfigurations- und Umgebungsdateien notwendig. Obwohl diese Vor- und Nachbereitungsschritte keine direkten Aktionen auf das Testobjekt ausüben, werden sie dennoch im weitesten Sinne als Aktionsschritte aufgefasst.

Einen besonderen Testschritt bildet die Anweisung zum Anhalten der Testdurchführung. Da der Benutzer hin und wieder auf Ergebnisse und Ausgaben des Testobjektes eine bestimmte Zeit lang warten muss (Messung abwarten), kann die Anweisung zum Warten als ein eigenständiger Testschritt mit dem einzigen Aktionsschritt „Warten“ angesehen werden.

Definition Testpaket: *Ein Testpaket ist eine Gruppe von unter einem bestimmten Aspekt zusammengehörenden Testfällen zur Überprüfung ein und des selben Testobjektes. Alle Testfälle für einen Anwendungsfall zusammen bilden ein Testpaket.*

Definition Software-Kriterium (IEEE): *Eine unterscheidbare Eigenschaft eines Softwareobjektes. (z.B. Performanz, Portabilität, Funktionalität)*

Definition Testen (IEEE): *Der Prozess zur Analyse eines Softwareobjektes mit dem Ziel, die Unterschiede zwischen existierenden und gewünschten Zuständen zu finden (Fehler) und die Kriterien des Softwareobjektes zu bewerten.*

Definition Testsystem: [2, S.364] *Ein Testsystem unterstützt im wesentlichen folgende Funktionen: Benutzerschnittstelle, Unterstützung beim Konstruieren von Testfällen, Testausgabe, Testergebnisvergleich, Testfallsteuerung und C1-Messung.*

Die C1-Messung wird durch unser Testsystem nicht vorgenommen. Dafür kommt ein externes Werkzeug zur Instrumentierung des Quellcodes und zur Analyse der funktionalen Überdeckung zum Einsatz (Kapitel 5.1.1). Alle anderen Aufgaben eines Testsystems wurden in unserer Arbeit jedoch weitgehend umgesetzt.

1.2 Motivation

Das XCTL-System unterliegt ständigen Verbesserungen und Erweiterungen durch die Studenten des Projektseminars „Software-Sanierung“. Die Aktualisierung einer Programmkomponente könnte dabei Veränderungen des Verhaltens in anderen Komponenten hervor rufen, die nach Einspielen der neuen Version zunächst unentdeckt blieben. Eine abweichende Funktionalität festzustellen ist sicherlich schon sehr schwierig, aber eine Version mit dem ursprünglichen Verhalten aus dem CVS wiederherzustellen, um festzustellen, welche Modifikation dazu führte, ist eine unzumutbare Aufgabe.

Deshalb ist es sinnvoll, schon vor dem Einspielen einer neuen Version in das CVS-Repository, alle Programmkomponenten auf ihr Verhalten hin zu überprüfen. In der Softwaretechnik spricht man hierbei von einem „Regressionstest“. Die wiederholten Testläufe nach jeder Quellcodemodifikation sollen dabei sicher stellen, dass das Programm jede Funktion noch genauso wie vor seiner Veränderung erfüllt. Dabei muss jedoch beachtet werden, dass die Bedingungen, Eingaben und Schritte der Testfälle nach ihrer Definition nicht mehr verändert werden dürfen, um einen Vergleich der Ausgaben mit denen aus früheren Versionen zu ermöglichen.

Für einen Regressionstest dürfen jedoch ohne Weiteres neue Testfälle bzw. Testsequenzen definiert und dem Testpaket hinzugefügt werden. Dies ist zum Beispiel immer dann notwendig, wenn eine neu entwickelte Programmfunktionalität wie die „Automatische Justage“ dem Gesamtsystem hinzugefügt wird. Neue Testfälle zur Überprüfung der hinzugekommenen Funktionaliäten sind zu spezifizieren und nach jeder zukünftigen Versionsaktualisierung im Regressionstest zusätzlich durchzuführen. Werden im Zuge des Reengineerings einzelne Komponenten in ihrem Ein- bzw. Ausgabeverhalten gegenüber älteren Versionen verändert, müssen die zugehörigen Testfälle ebenfalls an die

neue Situation angepasst werden. Beispielsweise waren alte Testsequenzen zur Komponente „Diffraktometrie“ (LineScan) ab einer bestimmten Version aufgrund des veränderten Verhaltens nach Überarbeitung dieser Komponente nicht mehr erfolgreich durchführbar, da ihre Benutzerschnittstelle erheblich verändert wurde. Die Entwickler, welche eine jeweilige Programmkomponente im Zuge des Projektes überarbeiten, sind verantwortlich für die Wartung und Aktualisierung aller zugehörigen Testfälle, die diese Komponente betreffen! Eine übersichtliche Verwaltung und Wartung der Testfälle ist daher für den Erfolg von regelmäßigen Regressionstests extrem wichtig.

Da ein Regressionstest durch ständige Versionsupdates sehr häufig anfällt und in Abhängigkeit der Testfallanzahl und -komplexität ziemlich aufwendig werden kann, bietet sich eine Automatisierung dieses Vorgangs an. Dafür spricht auch die statische Natur der Eingaben und erwarteten Ausgaben. Skripte und Konfigurationsdateien, sowie Solldateien und Referenzgrafiken können für jeden Regressionstest wiederverwendet werden, solange das Verhalten der jeweiligen Komponenten nicht bewusst verändert wurde.

Ziel dieser Diplomarbeit ist die Entwicklung eines Testsystems zur Unterstützung bei der Durchführung von Regressionstests durch weitestgehende Automatisierung. Da zukünftige Regressionstests von verschiedenen Teilnehmern des Softwaresanierungs-Projektes durchgeführt werden, legen wir bei der Umsetzung dieser Aufgabe besonders großen Wert auf Benutzerfreundlichkeit und eine transparente Testfallverwaltung.

Kapitel 2

Die Teststrategie

2.1 Vorüberlegungen

Testen ist eine wesentliche Maßnahme zur Qualitätsicherung und findet in allen Phasen der Softwareentwicklung mit unterschiedlichen Zielen statt. Grundsätzlich werden dabei die Testphasen *Modultest* (Komponententest), *Integrationstest*, *Funktionstest*, *Systemtest*, *Abnahmetest* und *Installationstest* unterschieden (siehe Abbildung 2.1).

Testfälle für Module werden entworfen, um die Teilfunktionen eines Gesamtsystems isoliert voneinander zu überprüfen. Die Testobjekte sind in diesem Fall Codesegmente, wie ein oder mehrere zusammengehörige Funktionen. Beim Integrationstest wird das Verhalten nach Einbindung dieser Komponenten in das Gesamtsystem untersucht.

Im Funktionstest versucht man Unstimmigkeiten zwischen dem Programm und seiner externen Spezifikation aufzudecken. Eine externe Spezifikation ist eine genaue Beschreibung des Programmverhaltens aus der Sicht der Umgebung bzw. des Benutzers. Er ist gewöhnlich blackbox-orientiert und erfasst alle Funktionen des Gesamtsystems. Die Testfälle entstehen aus der Analyse der Spezifikationen des Testobjektes.

Im Gegensatz dazu steht der Systemtest als Testverfahren, welches das Programm mit seiner ursprünglichen Leistungsbeschreibung vergleichen soll. Dabei können die Testfälle nicht aus den Spezifikationen gewonnen werden. Systemtestfälle werden durch die Analyse der Leistungsbeschreibung entworfen und mit Hilfe der Analyse der Benutzerdokumentation formuliert. Leistungsbescheibungen legen fest, was und wie gut ein Programm eine Funktion leisten soll, jedoch nicht die Darstellungen dieser Programmfunktionen.

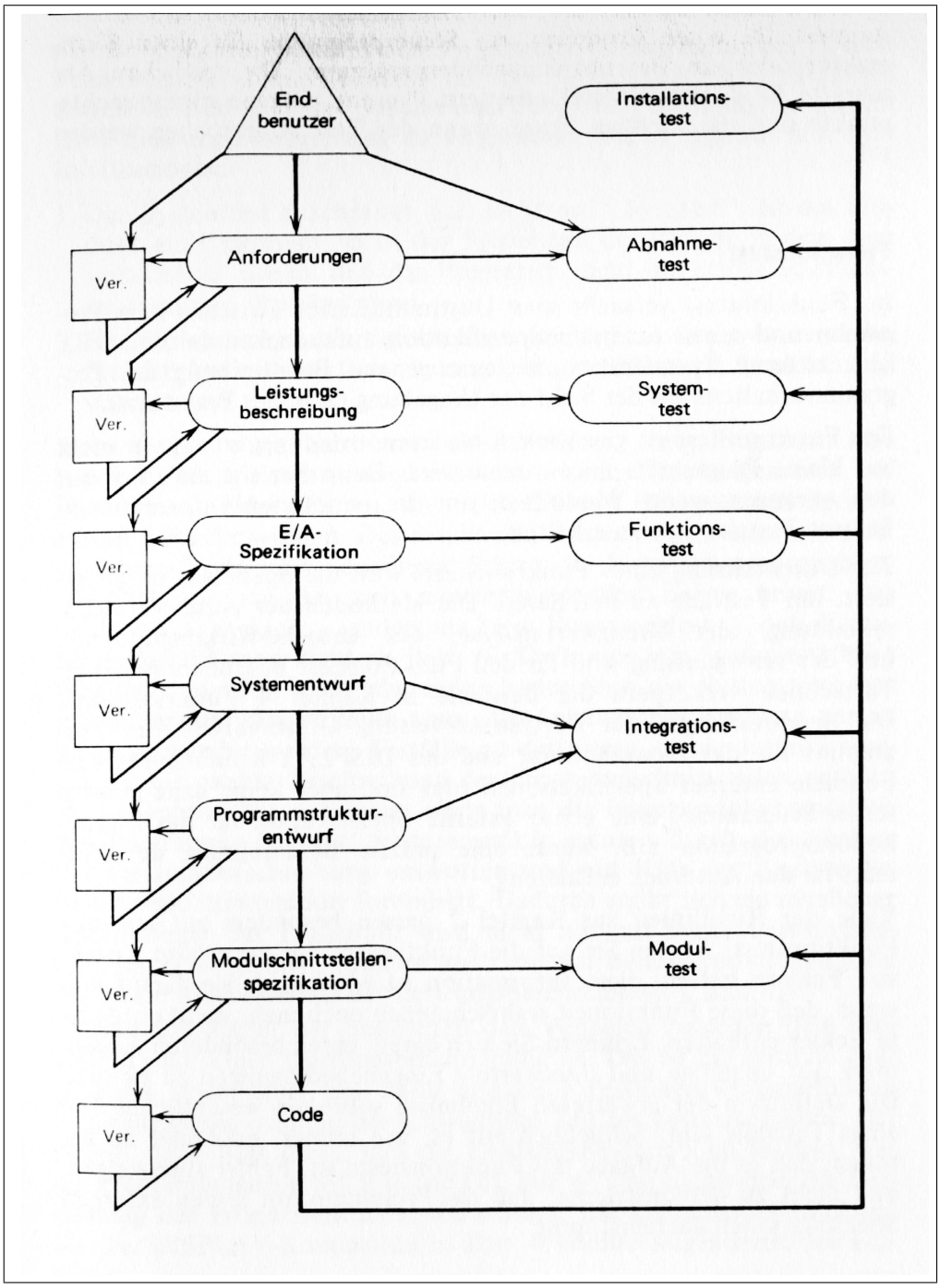


Abbildung 2.1: Die Korrespondenz zwischen Entwicklungs- und Testprozess (Quelle [3, S.107])

Zum Beispiel wird darin festgehalten, wann das Ergebnis einer bestimmten Funktion unter normaler Last zurückgegeben werden soll. Insgesamt werden 15 Kategorien von Systemtests unterschieden, die an dieser Stelle nicht näher erklärt werden sollen. Der interessierte Leser möge sich dazu in der Literatur [3, S.111] informieren. Ein Testobjekt kann hinsichtlich Vollständigkeit, Volumen, Last, Benutzerfreundlichkeit, Sicherheit, Leistung, Speicher, Konfiguration, Kompatibilität, Installation, Zuverlässigkeit, Fehlerbehandlung, Wartbarkeit, Dokumentation und Vorschriften untersucht werden. Diese Kategorien entsprechen den Software-Kriterien (siehe Begriffe, Abschnitt 1.1), unter deren Gesichtspunkten Testfälle entworfen und dem Test hinzugefügt werden können.

Abnahmetests sollen das Programm mit den ursprünglichen Anforderungen und den laufenden Bedürfnissen der Endbenutzer vergleichen. Die Testfälle werden dabei von den Endbenutzern selbst definiert und durchgeführt. Treten dabei keine Fehler auf, wird das Produkt akzeptiert.

Der letzte Testprozess ist der Installationstest und findet außerhalb des Entwicklungsprozesses statt, um Fehler bei der Einrichtung der Software auf der Zielplattform zu beseitigen. Die Testfälle werden von der Entwicklungsabteilung selbst erstellt und dem Produkt beigelegt, um nach einer Installation durchgeführt zu werden. Überprüft werden dabei Aspekte, wie beispielsweise die Existenz und Korrektheit der installierten Dateien, oder die Hardwareanforderungen an die Zielplattform.

Was ist nun ein Regressionstest? Der Regressionstest ist die Phase, die man nach einer funktionellen Verbesserung oder Reparatur eines Programmes durchführt. Man versucht festzustellen, ob die Änderungen das Programm in anderen Teilen in irgend einer Weise beeinflussen. Dabei lässt man gewöhnlich einige der vorher definierten Testfälle nochmals ablaufen. Regressionstests können damit allgemein in jeder Entwicklungsphase mit ihren jeweiligen Testfällen durchgeführt werden.

Unsere Teststrategie soll auf der Ebene des Gesamtsystems stattfinden. Die Aufgaben der Projektteilnehmer beschränken sich häufig auf die Bearbeitung einer bestimmten Teilmenge von Quellcode, die sich zu einem Anwendungsfall zuordnen läßt. Änderungen diese Komponenten sollen keine Auswirkungen auf andere haben. Da nicht eine bestimmte Eigenschaft, wie Performanz oder Belastung, sondern die Funktionalitäten des Steuerprogrammes gewährleistet werden soll, ist es sinnvoll, die Testfälle aus dem Funktionstest für die Zusammenstellung eines Regressionstestpaketes zu verwenden. Unser Projekt unterliegt keinem Forward-Engineering- sondern ei-

nem Reverse-Engineering-Prozess, womit solche funktionalen Testfälle noch nicht vorzufinden sind. Ein wesentlicher Teil dieser Arbeit besteht somit aus dem Entwurf von neuen funktionalen Testfällen, die den momentanen Zustand des XCTL-Systems erfassen und in allen zukünftigen Regressionstests durchgeführt werden müssen.

Grundsätzliches Ziel der oben beschriebenen Testmethoden ist es, durch wohl überlegte Testfälle, möglichst viele Fehler ausfindig zu machen und zu beseitigen. Das Ziel eines Regressionstestes ist hingegen ein anderes. Durch die Überprüfung von definierten Testfällen sollen Abweichungen der Funktionalitäten aus früheren Versionen aufgefunden werden. Das bedeutet jedoch, dass diese Testfälle in früheren Versionen des Testobjektes einwandfrei durchliefen. Umgekehrt müssen die Testfälle für einen Regressionstest demnach so entworfen werden, dass sie in der aktuellen Version zu keinen Fehlern führen.

Funktionstests sind sogenannte spezifikationsbezogene Tests, da die Testfälle direkt aus den Spezifikationen des Testobjektes abgeleitet werden. Ein solcher funktionaler Test, bei dem die interne Struktur des Programms verborgen bleibt, wird in der Fachsprache als Blackbox-Test bezeichnet und ist auch das umgesetzte Verfahren in unserem Projekt. Eingaben erhält das Programm über seine Benutzerschnittstellen zur Laufzeit. Ausgaben in Form von Dialogwerten, Messageboxen, Dateien oder Bildschirmgrafiken können mit entsprechenden Erwartungswerten verglichen werden.

Die Gewinnung bzw. Spezifikation von Testfällen für einen Whitebox-Test findet auf der Ebene des Quelltextes statt. Dazu ist die vollständige Kenntnis der Programminterna, wie die Beziehungen und Bedeutungen aller C++ Klassen notwendig. Der Quellcode der aktuellen Programmversion umfasst derzeit mehr als 30000 LOC, verteilt auf über 960 Funktionen und Methoden. Eine Whitebox-Methode ist bei einem Regressionstest auf Ebene des Gesamtsystems für ein Programm in dieser Größenordnung nicht sinnvoll und wird auch in der Literatur nicht empfohlen.

Zur Erfassung aller Funktionen ist ein Blackbox-Test jedoch oft unzureichend. In der Literatur [2, S.255] wird sogar von einem Greybox-Test gesprochen. Dabei soll weiterhin das Testobjekt nur über seine äußere Schnittstelle angesprochen werden. Die Testfälle müssen so gestaltet werden, dass sowohl die externen, aufrufbaren Funktionen wie auch alle internen „Ecken“, ausgeleuchtet werden.

Weitere Whitebox-Testfälle können erforderlich werden, wenn die angestrebte Überdeckung nicht auf Anhieb durch die Blackbox-Testfälle erzielt wird. Dieser recht vernünftige Vorschlag wurde von uns aufgegriffen und soll mit Hilfe der Instrumentierung (Kapitel 5.1.1) umgesetzt werden.

Das Ziel bei der Definition von ersten Testfällen für einen Regressionstest ist eine minimale Überdeckung aller über die Benutzerschnittstelle erreichbarer Programmfunktionen. Wenn man hier unüberlegt heran geht, kommen sehr schnell, sehr viele Testfälle zusammen, welche zudem keinerlei Ordnung unterliegen. Zur Strukturierung des Regressionstestpaketes hat sich die im Projektseminar herausgearbeitete Unterteilung des Steuerprogrammes nach seinen Anwendungsfällen (UseCases) angeboten. Die Testfälle werden für jeden Anwendungsfall entworfen, um alle darin enthaltenen Funktionen zu überprüfen. Die Wiederholung bzw. Redundanz von Tests bestimmter Funktionen wird damit innerhalb des Regressionstests weitgehend vermieden. Es sei hier schon bemerkt, dass es durchaus Funktionen gibt, die durch die Anwendungsfälle nicht erfasst werden. Funktionen zum Aufrufen der „About“-Dialogbox und der Hilfe oder sogar tote Funktionen sind dafür nur einige Beispiele. Im Buch „Software automatisch testen“ [1, S. 53] heißt es, dass ein Regressionstest alle Funktionalitäten erfasst, die sich als anwendbar erwiesen haben. Und genau das kann mit abgeleiteten Testfällen aus den Spezifikationen gewährleistet werden. Unbenutzte, unfertige oder sogar tote Funktionen dürfen in einem Regressionstest vernachlässigt werden. Die Diskussion zur Vollständigkeit der Testfälle wird im Kapitel 5.1 geführt.

Wenn man die Anwendungsfälle als eigenständige Module auffasst, entsprechen unsere entwickelten Testfälle denen eines Integrationstestes. Sie werden für Teilkomponenten (UseCases) konstruiert, jedoch in der Umgebung des Gesamtsystems durchgeführt. Ein großer Vorteil dieser Herangehensweise liegt auf der Hand. Die Testfall-Verwaltung findet wie alle anderen Entwicklerdokumente im Web-Repository geordnet nach dem jeweiligen Anwendungsfall statt, wodurch ihre Aktualisierung erheblich erleichert wird. Auf den Entwurf von wirksamen Testfällen für jeden Anwendungsfall wird genauer im Kapitel 3.3 eingegangen. Dieser und folgende Abschnitte dienen zunächst der Ausarbeitung und Vorstellung einer allgemeinen Vorgehensstrategie bei der Durchführung und Automatisierung von Regressionstests.

2.2 Anwendungsbereich

Umgebung und Anwendung des entwickelten Testsystems beeinflussen maßgeblich die Teststrategie. Regressionstests sollen durch die Projektmitarbeiter oder einen dedizierten Tester nach jedem Einspielen von Quelltext-Änderungen in das CVS-Repository durchgeführt werden können. Die Entwickler implementieren und überarbeiten dabei ihre Programmkomponenten häufig auf ihrem heimischen PC oder auf einem PC im Institut ohne Vorhandensein realer Motor- oder Detektorhardware.

Schon in seiner ursprünglichen Form bietet die Steuerungssoftware Funktionen zur Simulation von Hardware bzw. Controllern. Kommandos an die Hardware werden dabei abgefangen und berechnete Werte an das Programm wieder zurückgegeben. Ein Testlauf mit simulierter Hardware bringt ganz nebenbei einen weiteren großen Vorteil mit sich. Die empfindliche und teure Apparatur kann mit noch so obskuren Testfällen nicht zerstört werden. In der Simulation können beispielsweise Motorbewegungen um den Faktor 1000 beschleunigt werden. Man kann sich leicht vorstellen, dass dadurch die Gesamtdauer eines Regressionstests erheblich gesenkt werden kann.

Sinnvoll ist demnach der Entwurf von Testfällen, welche nur auf Komponenten der Umgebungssimulation agieren, so dass ein Regressionstest schon frühzeitig auf einem Entwickler-PC völlig unabhängig von angeschlossenen Peripheriegeräten durchgeführt werden kann. Die Fähigkeiten des Testsystems selbst sollen jedoch keinesfalls beschränkt werden. Jede beliebige Aktion auf dem Testobjekt, die ein Benutzer durchführen kann, soll auch mit dem Testsystem möglich sein. Beschränkt wird lediglich die Definitionsfreiheit der Testfälle. Es sind keine Aktionen auf Funktionen erlaubt, die nur mit realer Hardware durchführbar sind.

Die Güte der Testfälle bzw. des gesamten Regressionstests ist daher von der Realitätsnähe der Umgebungssimulation sehr abhängig. Alle Geräte, die innerhalb der Umgebungssimulation angesprochen werden können, werden im nachfolgenden Abschnitt hinsichtlich ihres Simulationsverhaltens diskutiert. Vorher soll noch untersucht werden, wie weit sich eine reale Umgebung von der simulierten unterscheidet, also weshalb ein Test innerhalb der Umgebungssimulation überhaupt sinnvoll ist.

Die Motorsimulation arbeitet auf der untersten Kommunikationsschicht der Steuerungssoftware. Sie nimmt Controllerbefehle wie `Put` und `Get` mit entsprechender Adressierung entgegen und gibt berechnete Resultate zurück, die denen der realen Hardware entsprechen. Auf diese Weise wird nur das Verhalten einer Controller-Karte simuliert, währenddessen das Ein- und Ausgabeverhalten zwischen Hard- und Software davon unbeeinflusst bleibt.

Detektoren werden über Ableitungen von C++ Klassen angesprochen, die ihrerseits Methoden zur Kommunikation mit der eigentlichen Hardware bereitstellen. Die Klassen der simulierten Detektoren stellen Methoden zur Kommunikation mit dem XCTL-System zur Verfügung, ohne dafür auf zusätzliche Hardware zuzugreifen. Testfälle, die den Quellcode für die Klassen realer Detektoren betreffen, sind für den Regressionstest nicht zulässig. Das ist für die Überprüfung der Funktionstüchtigkeit einzelner Komponenten auch nicht notwendig, da sich sehr viele Funktionen mit den simulierten Rückgabewerten ausführen lassen. Der viel umfangreichere Kern des Steuerprogramms, welcher für die Verarbeitung der Rückgabewerte verantwortlich ist, lässt sich weitgehend mit entsprechenden Testfällen untersuchen. Die Funktionalitäten realer Detektoren ändern sich im Gegensatz zum Softwarekern der Steuerungssoftware nur sehr selten bzw. überhaupt nicht. Häufige Quellcode-Veränderungen an den vorhandenen Detektor-Klassen sind damit nicht zu erwarten.

Unter diesen Umständen führen mit hoher Sicherheit erfolgreiche Testläufe innerhalb der Umgebungssimulation zu Programmversionen, die auch an den Messplätzen mit angeschlossener Hardware fehlerfrei arbeiten. Ein Feldtest unter realen Einsatzbedingungen darf bei allem Optimismus dennoch nicht vergessen werden. Auch wenn der Regressionstest alle Funktionalitäten erfolgreich absolviert hat, müssen bei der Auslieferung einer aktuellen Version an die Physik in einem Abnahmetest alle Funktionalitäten noch einmal unter realen Bedingungen getestet werden. Dazu können jedoch nicht die spezifizierten Testfälle dieser Arbeit herhalten. Die Spezifikation der Testfälle ist ausschließlich auf den Einsatz innerhalb der Umgebungssimulation abgestimmt, um die viel häufigeren Regressionstests an den Entwickler-PCs zu unterstützen und mögliche Fehlerquellen schon frühzeitig abzustellen. Alle Testdaten und Erwartungswerte der definierten Regressionstestfälle sind Rückgabewerte der simulierten Motoren und Detektoren.

2.3 Umgebungssimulation

2.3.1 Unterstützte Geräte

Für eine ausreichende Simulation der Peripherie müssen die wichtigsten Motor- und Detektortypen mit ihren Funktionen dem XCTL-System zugänglich sein. Die Umgebungssimulation wurde schon vom ursprünglichen Programmierer Herrn Damerow für solche Zwecke vorgesehen und im Laufe des Projektseminars von den Studenten weiter entwickelt. Um die zur Verfügung stehenden Geräte der Steuerungssoftware herauszuarbeiten, ist ein Blick in den Quellcode notwendig. Nur wenn man die Zusammenhänge zwischen den jeweiligen Hardware-Klassen versteht, erkennt man die Möglichkeiten und Grenzen der Umgebungssimulation.

Motoren Das XCTL-System akzeptiert der Version vom 20.01.2002 vier verschiedene Motortypen. Das sind im Einzelnen die Typen C-812ISA, C-812GPIB, C-832 und TMotor. Wie in Abbildung 2.2 zu erkennen, definiert zuletzt genannter Typ eine Oberklasse der restlichen Typen und stellt seinen Ableitungen grundsätzliche Funktionalitäten, wie die Berechnung der minimalen Schrittweite in Nutzereinheiten, zur Verfügung. Unbekannte Motoren aus den INI-Dateien der Steuerungssoftware werden automatisch zu Objekten dieses generischen Typs. Da dieser Typ virtuelle Methoden zur Positionierung der Motorstellung definiert, ohne dabei auf Hardware zuzugreifen, ist er generell zum Einsatz für Testfälle in der Umgebungssimulation geeignet.

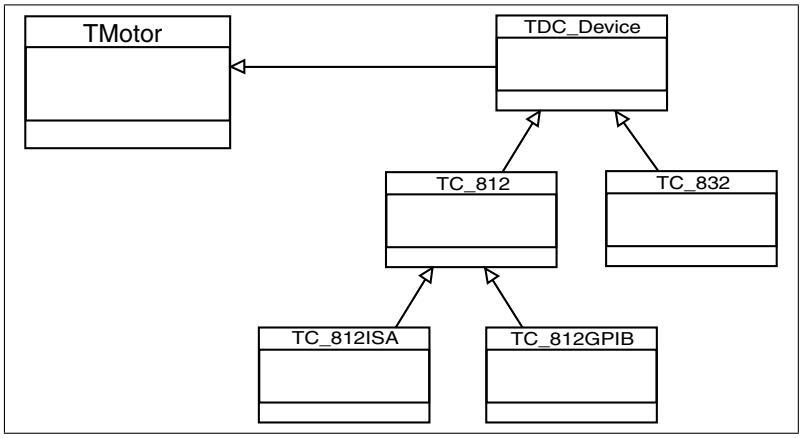


Abbildung 2.2: Vererbungshierarchie der Motorklassen

Die Methoden zur Positionierung und Ansteuerung der Motoren werden in den Ableitungen für C-812 und C-832 Antriebe überschrieben, um darin Steuerkommandos direkt an den angeschlossenen Controller zu schicken. Mit eingestellter Option in der INI-Datei kann dieser Vorgang abgefangen und simuliert werden. Im Rahmen einer Diplomarbeit ist dazu eine dynamische Bibliothek entstanden, die das Verhalten für C-812ISA und C-832 Motoren simuliert. Der Antrieb C-812GPIB wurde dabei nicht implementiert, da er in der Physik gegenwärtig nicht mehr verwendet wird. Beliebige Antriebe vom Typ C-812ISA und C-832 sind dagegen in ihrem Verhalten sehr realitätsnah nachgebildet und deshalb für den Einsatz in Testfällen innerhalb der Umgebungssimulation sehr gut geeignet.

Detektoren Der Aufbau der Detektor-Klassen ähnelt dem der Motoren. Das XCTL-System akzeptiert gegenwärtig die Typen Generic, Matrox, Radicon, Test, Encoder, Psd, Braun-Psd und Stoe-Psd. Unbekannte Typen in der INI-Datei werden automatisch zu Objekten der Klasse TDevice (siehe Abbildung 2.3). Die dargestellte Vererbungshierarchie und Benennung der Klassen entspricht dem Projektstand vor der Aktualisierung des Detektor-Subsystems.

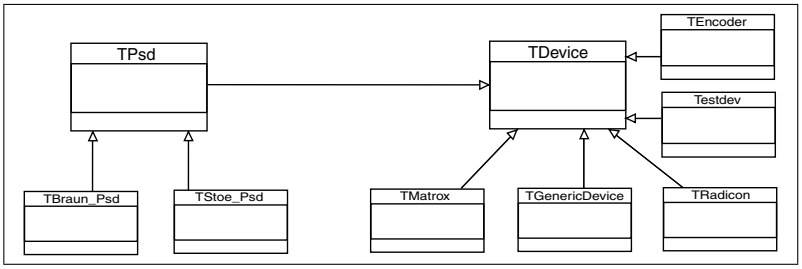


Abbildung 2.3: Vererbungshierarchie der Detektorklassen

Die beiden 0-dimensionalen Detektortypen Generic und Radicon benötigen zur Intensitätsermittlung zwingend angeschlossene Messhardware. Ihre Methoden kommunizieren direkt mit der Detektor-Hardware. Der Encoder erhält seine Intensitätswerte aus den Positionen eines gleichnamigen Antriebs. Innerhalb der Umgebungssimulation ist er damit zwar einsetzbar, aber da uns ein sinnvoller Anwendungsfall nicht bekannt ist, wird er in den definierten Testfällen nicht weiter vorkommen. Alle Typen bzw. Klassen sind Ableitungen von TDevice, welche analog zur Klasse TMotor grundsätzliche Funktionen für jeden Detektor definiert. Objekte dieser Klasse liefern berechnete Intensitätswerte in Abhängigkeit der Positionen von Omega und Psi ohne

einen Zugriff auf angeschlossene Detektor-Hardware und sind daher für den Einsatz in Testfällen mit 0-dimensionalen Detektoren im Rahmen der Umgebungssimulation geeignet. Der Detektortyp Test wurde während der Projektarbeit ausschließlich für Simulationszwecke entwickelt. Seine Methoden zur Simulation der Intensitätswerte eines 0-dimensionalen Detektors in Abhängigkeit von drei Bewegungsachsen sind vollkommen unabhängig von peripheren Messgeräten, womit er sich für den Einsatz in Testfällen innerhalb der Umgebungssimulation hervorragend eignet.

Die Klasse für Detektoren vom Typ Psd enthält eine virtuelle Methode, die eine Intensitätsbestimmung in Abhängigkeit der Position von Theta simuliert. Die beiden Ableitungstypen Stoe-Psd und Braun-Psd überschreiben diese Methode und greifen direkt auf die Detektorhardware zu. Für den Einsatz in Testfällen mit 1-dimensionalen Detektoren innerhalb der Umgebungssimulation ist damit nur der Detektor vom Typ Psd (Klasse TPsd) geeignet. Der CCD-Detektortyp Matrox befindet sich noch in der Entwicklung bzw. Einbindung in das Steuerprogramm. Sein Quellcode ist in der aktuellen Programmversion auskommentiert, womit dieser Detektortyp für den Einsatz in Testfällen nicht zur Verfügung steht.

2.3.2 Bewertung

Motoren Die Motorensimulation hat sich, Dank der Arbeit eines Projektmitarbeiters, sehr stark verbessert. Größtes Manko der ursprünglichen Simulation von Herrn Damerow (Typ TMotor) ist die Bewegung ohne Zeitverlust und die Ignoranz gegenüber den Motorkommandos, die vom XCTL-Programm erzeugt werden. Jede Sollposition wird nach dem Aufruf der zuständigen Methode sofort erreicht. Damit sind Testfälle unmöglich, die den Fahrbetrieb der Motoren betreffen. Der simulierte Zeitverlust der entwickelten Motorsimulation ermöglicht dagegen die Definition von Testfällen für Funktionen, wie den „Continuous Scan“, die ausschliesslich in diesem Bewegungsmodus arbeiten. Die Beschleunigung und Abbremsung entspricht sehr exakt dem Verhalten realer Antriebe (siehe Diplomarbeit Lützkendorf [10]).

Dennoch haben einige Parameter auch in der neuen Motorensimulation keine Auswirkung auf das simulierte Verhalten. Davon sind hauptsächlich Einstellungen zur Stromversorgung der Antriebe betroffen. Alle wichtigen Parameter, die das Verhalten der Antriebe direkt beeinflussen und auf das Steuerprogramm zurück wirken, werden auch in der Simulation ausgewertet. Im Kapitel 3.6 wird bei der Konstruktion von Konfigurationsdateien die Bedeutung der Motor-Parameter innerhalb der Umgebungssimulation ausführlich diskutiert.

Detektoren Der 0-dimensionale Testdetektor vom Typ TDevice berechnet nur in Abhängigkeit der Antriebe Omega und Psi Intensitätswerte, die nicht auf real gemessenen Werten beruhen. Zusätzlich wird dabei ein zufälliger Anteil mit eingerechnet, um den Eindruck einer realen Messaparatur zu verstärken. Ergebnis ist ein, von der Motorposition abhängiger, zufällig schwankender Messwert, dessen Maximum beim Omegawinkelwert 0 liegt und der nach beiden Seiten hin abnimmt. Der berechnete Intensitätswert ist unabhängig von der eingestellten Messzeit und Impulsbegrenzung. In der Realität würde bei Erreichen der maximalen Impulszahl vor Ablauf der Messzeit der Intensitätswert auf die vorgegebene Messzeit hochgerechnet und ein neues Messintervall bei Null begonnen werden. Aufgrund fehlender Controller- bzw. Detektorhardware können innerhalb der Umgebungssimulation weder Soundausgaben veranlasst noch überprüft werden. Endgeräte zur Soundausgabe befinden sich direkt auf dem Controller und dienen zur akustischen Information über die aktuelle Intensitätsstärke.

Auch dem nachträglich entwickelten 0-dimensionalen Testdetektor vom Typ Test fehlt es an Funktionalitäten zur Soundausgabe. Messzeit und Impulsbeschränkung bleiben zur Berechnung der Intensität genauso unbeachtet. Trotzdem ist dieser Detektortyp für den Einsatz in Testfällen des Regressionstests unverzichtbar, da er seine Intensitäten in Abhängigkeit der Antriebe DF, TL und Kollimator berechnet. Die Grundlage der Ergebnisse bildet hierbei jedoch nicht eine Formel, sondern eine Datenbasis, die unter realen Bedingungen im Labor aufgezeichnet wurde und zur Laufzeit ausgelesen wird. Ein zufälliger Anteil soll auch hier den Eindruck einer realen Messaparatur hervorrufen. Die Abhängigkeit der Messwerte von den drei wesentlichen Antrieben eines Topographie-Arbeitsplatzes eröffnet zum Beispiel die Möglichkeit, sinnvolle Testfälle für die Automatische Justage zu definieren. Programmintern werden die Antriebe DF und TL, wie Omega und Tilt behandelt, weshalb die Datenbasis auch in Testfällen zur Diffraktometrie/Reflektometrie benutzt werden kann. Obwohl die Abtastung der realen Messprobe in recht groben Schritten erfolgte, ist es mit den aufgezeichneten Werten der Datei `TESTDEV.DAT` dennoch möglich, sehr viele Programmfunktionalitäten des XCTL-Programms, ohne Anwesenheit realer Hardware eines 0-dimensionalen Detektors, durchzuführen.

Der 1-dimensionale Testdetektor vom Typ Psd liegt mittlerweile nicht mehr in seiner ursprünglichen Implementierung vor. Er liefert, unabhängig von der Anzahl der Kanäle, ein zufälliges Spektrum mit einem Peak (maximale Intensität) auf der Mitte. Die Funktionalität der Simulationsmethode wurde so erweitert, dass nun zusätzlich die Psd-Intensitäten von der bereits

abgelaufenen Messzeit und die Peakpositionen vom Antrieb Theta abhängen. Zur Definiton von Testfällen für den Anwendungsfall „AreaScan“ ist die Simulation damit völlig ausreichend. Der Test-Detektortyp unterscheidet jedoch nicht zwischen den beiden Anzeigearten Energie- und Impuls-Spektrum. In der Realität kann das Impulsspektrum zum Teil starken Schwankungen unterliegen, da es vom untersuchten Substrat abhängig ist. Dagegen wird das Energiespektrum direkt aus der Strahlungsquelle bestimmt und hat daher ihr Maximum immer bei den gleichen Kanälen. Die Unterscheidung der zwei Anzeigearten wird bislang nur in den Methoden des Braun-Psd's vorgenommen.

Fazit Der Freiheitsgrad der möglichen Testfälle innerhalb der Umgebungssimulation wird durch die Funktionen der vorgestellten drei Testdetektoren und der beiden Motortypen eingeschränkt. Die Konfiguration des Radicon-Zählers, die sich aus der Dialogbox zur Detektor-Einstellung aufrufen läßt, ist damit beispielsweise nicht testbar. Alle Testdaten müssen zudem so definiert werden, dass sie auf der simulierten Hardware zu sinnvollen und auswertbaren Ergebnissen führen. Der 0-dimensionale Testdetektor liefert zum Beispiel auf Grundlage seiner Messdaten (`TESTDEV.DAT`) nur in bestimmten Motorbereichen Werte, die sich zur Messung der Halbwertsbreite (HWB) eignen. Eine HWB-Messung kann nur auf einem Peak durchgeführt werden. Die Intensitätswerte in unmittelbarer Umgebung des Omega-Antriebs müssen monoton abfallen. Solche Intensitätsmaxima sind nur innerhalb des vorliegenden Messdatenbereichs auffindbar, weshalb die Gesamtzahl der möglichen HWB-Werte auf die Menge aller Peaks aus dieser Datenbasis begrenzt ist. Beschränkungen der Möglichkeiten und Hinweise bei der Spezifizierung von Testfällen wird für jeden Anwendungsfall in den jeweiligen Abschnitten unter 3.3.3 beschrieben.

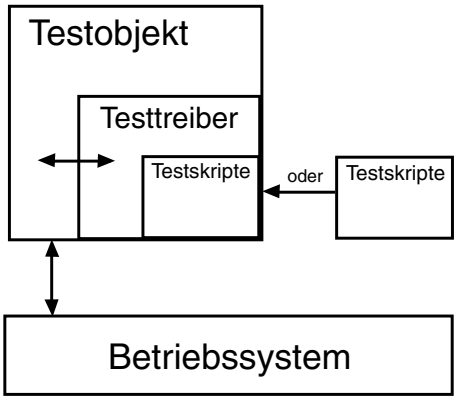
Zur Spezifikation und Durchführung von erschöpfenden Testfällen ist die Umgebungssimulation in ihrer derzeitigen Form völlig ausreichend. Nur wenige Funktionen des Steuerprogrammes sind mit den zur Verfügung stehenden Geräten nicht durchführbar. Apparatur-Ansteuerung und Verarbeitung der Rückgabewerte können, als grundsätzliche Aufgaben des Programmkerns, auf Korrektheit überprüft werden. In zukünftiger Projektarbeit könnten weitere Attribute, wie Messzeit und Intensitätsbegrenzung, den Detektor-Simulatoren hinzugefügt werden. Der 1-dimensionale Testdetektor könnte, ähnlich wie der 0-dimensionale, um eine Messdatendatei erweitert werden, auf die in Abhängigkeit der Motorpositionen Omega und Theta zugegriffen wird. Die Möglichkeiten zur Definition von Testfällen wird damit zwar nicht erhöht, wohl aber ihre Nähe zu realen Messkurven.

2.4 Automatisierung

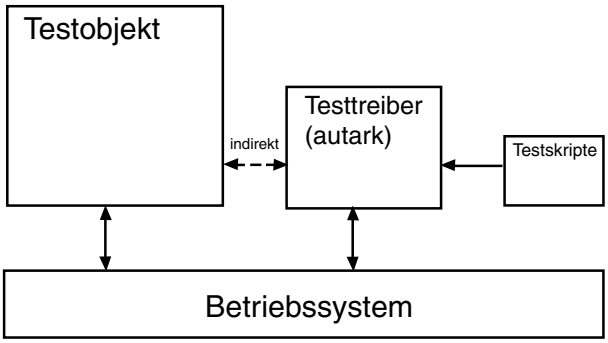
2.4.1 Varianten

Als BlackBox-Testfälle, die aus den Spezifikations-Dokumenten des Testobjektes aus der Sicht eines Benutzers entstanden sind, soll die Automatisierung ihrer Durchführung ebenfalls auf Ebene der grafischen Benutzeroberfläche stattfinden. Dafür bieten sich im allgemeinen die folgenden zwei unterschiedlichen Strategien an:

1. *Testtreiber als Quellcode-Erweiterung:* Das Testobjekt kann durch Erweiterung seines Quellcodes um eine Test-Dialogbox ergänzt werden, um den Zugriff auf die Benutzerschnittstelle (Fenster, Dialogboxen) zu vereinfachen. Alle Fensteraufrufe und Eingaben erfolgen programmintern, womit sich das Testobjekt aus sich selbst heraus testet. Die Testsequenzen können dabei extern bereitliegen oder im Zuge der Programmerstellung mit hineinkompiliert werden.



2. *Autarker Testtreiber:* Ein unabhängiges System prüft das Testobjekt, welches dazu alle Testdaten den Benutzerschnittstellen über das Betriebssystem zuführt. Das Programm wird zum Test sozusagen ferngesteuert. Die Testsequenzen zur Beschreibung der durchzuführenden Schritte werden dazu vom Testsystem interpretiert und als Windows-Nachrichten auf dem Betriebsystem ausgeführt.



In beiden Varianten können die Ausgaben zunächst gesammelt werden, um sie erst am Ende des Testlaufs mit Erwartungswerten zu vergleichen. In einigen Fällen ist es jedoch sinnvoller, schon zur Zeit der Testdurchführung, die Ausgaben bzw. Reaktionen des untersuchten Programmes auszuwerten.

Wir haben uns für eine testfallbegleitende Auswertung entschieden, da die Testschritte im allgemeinen voneinander abhängig sind und damit die Korrektheit des Ergebnisses eines Testschrittes vor der Ausführung des nächsten sicherzustellen ist. Nicht automatisierbare Auswertungsschritte erfordern sogar den Eingriff des Testers innerhalb einer Sequenz. Herr Memon beschreibt in seiner Dissertation [4, S.6/7], dass dieses Vorgehen bei einer Testmethode über die grafische Benutzerschnittstelle (GUI) sogar notwendig ist. Damit entsprechen die Skripte strukturell den Testfall-Dokumenten im Web-Repository. Eingaben führen zu Ausgaben, die unmittelbar zu vergleichen sind.

Die meisten kommerziellen Programme zur Testautomatisierung arbeiten mit einem *Capture and Replay*-Verfahren. Dabei werden alle GUI-Aktivitäten auf einer zu testenden Windows-Applikation während der manuellen Durchführung eines Testfalles zunächst aufgezeichnet. Die Eingaben per Maus oder Tastatur können meistens durch zusätzliche Aktionen mit entsprechenden Skriptkommandos erweitert werden. Sogenannte Checkpoints entsprechen den Auswertungsschritten und können an beliebigen Stellen eingefügt werden. Der bekannteste Vertreter dieser Werkzeuge ist das Programm *WinRunner* von *Mercury Interactive* [24].

Es kann in zwei verschiedenen Modi arbeiten. Der *kontextsensitive* Modus ist unabhängig von der physischen Lage der Oberflächenelemente auf dem Bildschirm, da alle GUI-Elemente als logische Objekte erkannt werden (Windows-Conrols: Buttons, Listboxen etc.). Dieser Modus eignet sich hervorragend zum Testen von Masken-orientierten Anwendungen. Der Modus

Analog hingegen zeichnet alle Eingaben (Mausklicks) über physische Koordinaten auf. Dieses pixelbasierte Vorgehen eignet sich daher besonders zum Testen von Zeichenprogrammen.

Testwerkzeuge, die nur den analogen Modus anbieten, sind für unser Testobjekt nicht geeignet, weil schon geringe Veränderungen an der Positionierung der GUI-Elemente zu Problemen führen. Leider waren wir bei der Suche nach kostengünstigen Testwerkzeugen, die den kontextsensitiven Modus anbieten, nicht erfolgreich. Schon frühzeitig fiel deshalb die Entscheidung zugunsten der Implementierung eines eigenen Testsystems aus.

Für die Realisierung eines *Testtreibers als Erweiterung des Quellcodes* spricht der geringe Implementierungsaufwand, da man auf alle Elemente der Benutzerschnittstellen einen einfachen, programminternen Zugriff hat. Wenn das Testsystem im Programm selbst vorliegt, sind alle Steuerelemente über die internen Datenstrukturen ansprechbar. Der größte Nachteil dieser Testmethodik liegt jedoch in ihrer Unflexibilität. Für jeden Test einer aktuellen Version müsste der gesamte Testrahmen zuammen mit dem eigentlichen Testobjekt kompiliert werden. Auch dynamisches Linken der Testkomponente als DLL würde Abhängigkeiten vom Quelltext des XCTL-Systems nach sich ziehen. Spuren dieser Testkomponente würden so oder so im Endprodukt übrig bleiben. Das Testsystem wäre ein fester Bestandteil der Programmquellen und müsste im selben Modul des CVS-Repository's gewartet werden. Für den Regressionstest wollen wir jedoch so unabhängig wie möglich vom Quellcode des XCTL-Systems bleiben, weshalb diese Variante auszuschliessen ist.

Wir haben uns für die Realisierung eines autarken Testsystems mit manueller Entwicklung von Testskripten auf der Grundlage vorliegender Quelltexte entschieden. Sie ist ein Kompromiss aus unseren Programmierkenntnissen und der Zweckdienlichkeit des zu entwickelnden Testsystems. Obwohl der Aufwand zur Entwicklung eines autarken Testsystems erheblich ist, bringt das Ergebnis beträchtliche Vorteile mit sich. Durch den uneingeschränkten Zugriff auf die Betriebssystemumgebung, wie das Dateisystem und die Prozessverwaltung, ist der Einsatz eines autarken Testsystems sehr flexibel.

Mit den Möglichkeiten der Windows-Nachrichtenverwaltung gibt es kaum Grenzen für das Testsystem. Eingaben in die Dialogboxen können vorgenommen und Rückgabewerte wieder ausgelesen werden. Gleichzeitig können die erzeugten Ausgabedateien auf ihre Inhalte mit vorgegebenen Solldateien verglichen werden. Seine unabhängige Arbeitsweise lässt das Testsystem von fehlerbedingten Abstürzen des Testobjektes weitestgehend unberührt. In den

angelegten Protokollen kann man leicht nach Ursachen für den Zusammenbruch suchen. Außerdem können vom Testsystem Maßnahmen eingeleitet werden, um das Testobjekt wieder in seinen ursprünglichen Zustand zu versetzen. Es können sogar Testfälle definiert werden, in denen das Testobjekt mehrfach gestartet und beendet wird. Der Einsatz einer Skriptsprache legt dabei nicht das Verfahren zur Gewinnung eines Testskriptes fest. Mit einer gemeinsamen Grundlage, welche durch die festgelegte Skriptsprache geschaffen wird, können Testskripte sowohl manuell, als auch automatisiert entwickelt werden.

Nachteilig ist die Umständlichkeit bei der Entwicklung von Testskripten auf Grundlage der vorliegenden Quellen und RC-Dateien. Die Testskripte können nicht, wie beim *Capture and Replay*-Verfahren, direkt auf der zu testenden Anwendung durch Benutzung der Steuerelemente automatisch erstellt werden. Vielmehr müssen Skripte auf Grundlage der Programm-Ressourcen manuell erstellt werden. Unser Testsystem soll daher dem Entwickler zur Bewältigung dieser Aufgabe Unterstützung anbieten (siehe Abschnitt 3.8).

Neben der Ausführung von Testschritten auf der grafischen Benutzeroberfläche wird ein Verfahren gefordert, welches zusätzlich den Vergleich von Ausgabedateien der Testapplikation automatisiert. Wegen der extern vorliegenden Dateien ist die Entwicklung eines autarken Programmes für diese Aufgabe sinnvoll. Ein flexibler und vollautomatischer Vergleich einer Datei mit einer Referenzdatei ist eine leicht zu realisierende Aufgabe und wird mit dem Programm *DataDiff* (siehe Abschnitt 3.7.1) umgesetzt.

2.4.2 Schwierigkeiten

Die Implementierung eines eigenen Testrahmens zur Umsetzung eines automatisierten Regressionstests bringt dennoch einige Schwierigkeiten mit sich. Einige Realisierungsprobleme sind dadurch begründet, dass uns für die Bearbeitung der Aufgabenstellung nur ein begrenzter Zeitrahmen zur Verfügung steht. Auch unzureichende Erfahrungen in der Windowsprogrammierung schränken unsere Möglichkeiten ein. In der Vorarbeit stand daher eine Aufwand/Nutzen-Abschätzung. Es musste überlegt werden, wie wir das Ziel der Automatisierung von Regressionstests möglichst allgemein und umfassend umsetzen können, ohne dabei zu viel Arbeit und Zeit für Detailprobleme zu verschwenden. Die angesprochenen Schwierigkeiten betreffen Testschritte bzw. Auswertungsschritte, die sich mit den Möglichkeiten unseres Testsystems nicht automatisieren lassen. Die Automatisierung dieser Schritte ist

nicht generell unmöglich. Das Testsystem müsste dazu „nur“ um die notwendigen Funktionen erweitert werden. Da der Aufwand zur Realisierung solcher Funktionen in keinem Verhältnis zum seinem Nutzen steht, haben wir uns dafür entschieden, stattdessen interaktive Eingriffe des Testers während der Ausführung eines Skriptes jederzeit zuzulassen. Nicht automatisierbare Aktionsschritte und Auswertungsschritte werden zur Laufzeit der Testsequenzen manuell durchgeführt. Das Testsystem stellt dafür Funktionen bereit, die einen solchen Dialog mit dem Tester ermöglichen.

1. Problemkategorie: Im Testsystem nicht automatisierbar

Mausaktivitäten eines Benutzers auf den Messkurven-Fenstern, wie sie beispielsweise im Anwendungsfall „AreaScan“ bei der Datenerhebung vorkommen, lassen sich nur sehr schwer fernsteuern. Hierbei stößt man zunächst auf die schon angesprochene Problematik der Pixelabhängigkeit. Die Fenster müssen dafür immer an der gleichen Stelle mit einer festen Rahmengröße erscheinen, um Mausaktivitäten an ihnen durchführen zu können. Schwierig wäre die Definition und Interpretation von geeigneten Skriptbefehlen zur Positionierung der Maus und Auslösung von Aktivitäten (Maustasten) in bestimmten Fenstern. Einige kommerzielle Werkzeuge ermöglichen zwar die Aufzeichnung von Mausbewegungen und -aktionen, da wir jedoch ein solches „Capture and Replay“-Werkzeug nicht einsetzen, müssen wir auf eine Automatisierung solcher mausintensiven Testschritte verzichten. Zur Laufzeit wird der Testfall unterbrochen, um die Mausaktionen vom Tester manuell vornehmen zu lassen und anschließend fortzufahren.

Viele Ausgaben, wie die Darstellung von Intensitätswerten und Messkurven erscheinen in Form von Bildschirmgrafiken. Ein approximativer Vergleich wäre hierbei nur mit Techniken der Bildverarbeitung möglich. Auch für solche Aufgaben existieren kommerzielle Werkzeuge, die aus bereits erwähnten Argumenten nicht eingesetzt werden sollen. Die Implementierung eines eigenen Programmes zum Bitmapvergleich würde jedoch den Rahmen dieser Diplomarbeit bei weitem übersteigen, weshalb solche Auswertungsschritte in die Kategorie „nicht automatisierbar“ fallen. Zur Auswertung dieser Ergebnisse wird der Testfall unterbrochen, um den Tester zum manuellen Vergleich mit einer Referenzgrafik aufzufordern.

Eine weitere Kategorie von Schwierigkeiten bei der Realisierung des Testverfahrens umfasst allgemeine Probleme, die durch das Ein- und Ausgabeverhalten des zu testenden Objektes selbst verursacht werden.

2. Problemkategorie: Ein-/Ausgabeverhalten des Testobjekts

Sehr oft folgen Ausgaben und Ereignisse des XCTL-Systems den Eingaben zeitlich versetzt. Motorenbewegungen oder Berechnungen müssen abgewartet werden. Zur Lösung dieses Problems bieten sich zwei Möglichkeiten an:

1. Der zu untersuchende Wert wird ständig abgefragt, bis sich der erwartete einstellt. Diese „Polling"-Methode muss jedoch auch zeitlich begrenzt sein, damit im Falle eines Fehlers die Testdurchführung fortgesetzt bzw. abgebrochen werden kann.

2. Wenn die Dauer bis zum Einstellen des erwarteten Wertes bekannt ist, genügt die Abfrage nach dieser Zeitspanne. Es ist ein Fehler, wenn sich dieser Wert bzw. das Ereignis nach der definierten Zeitspanne nicht eingestellt hat.

Diese Vorgehensweisen sind notwendig, weil es dem XCTL-System nicht möglich ist, dem Testsystem mitzuteilen, wann der zu untersuchende Wert überprüft werden kann. Das XCTL-System hat keine Kenntnis von der Existenz eines Testsystem. Beide Methoden finden ihre Verwendung in den definierten Testfällen. Der Auswertungsschritt zum Warten auf Fenster, Dialogboxen und Messageboxen findet auf der unteren Skriptsprachebene (dazu später mehr) mittels Polling statt. Sofort nach ihrem Erscheinen ist der Auswertungsschritt erfüllt und der Testfall wird fortgesetzt. Wenn nichts passiert, wird nach Ablauf einer Maximalzeit der Testfall abgebrochen, da das geforderte Ereignis nicht eingetreten ist. Der Testschritt zum Abwarten einer bestimmten Zeitspanne hingegen kann von beliebigen Auswertungsschritten gefolgt werden. Innerhalb der Wartezeit kann die Ausführung der Testsequenz nicht abgebrochen werden. Diese Variante kommt in allen definierten Testfällen zum Einsatz, in denen verzögerte Ergebnisausgaben zu vergleichen sind. Beispielsweise kann die eingestellte Motorposition nach einer konstanten Zeitspanne abgefragt werden, wenn Geschwindigkeit und Strecke bekannt sind. Im allgemeinen ist somit die Durchführung eines automatisierten Testlaufs sehr abhängig von den Ausführungszeiten des letzten Aktionsschrittes eines Testschrittes. Daher müssen für die Spezifizierung neuer Testfälle alle Zeiten vom letzten Aktionsschritt bis zum Einstellen eines auswertbaren Ergebnisses ermittelt und dokumentiert werden.

Im Laufe der Realisierung des Testsystems ist uns ein weiteres Problem aufgefallen. Das Design der Steuerungssoftware macht es leider nicht möglich, an Ausgaben ihrer Statuszeile heranzukommen. Alle Status-Informationen werden zur Laufzeit in ein Bitmap umgewandelt und auf dieser Zeile als

Grafik ausgegeben. Wie im Abschnitt 2.5.2 erklärt, können Ausgaben des Testobjektes von außen nur über Windows-Nachrichten ausgelesen werden. Wenn die Ausgaben der zu testenden Software nicht über Standardfunktionen realisiert werden, kann das Testsystem darauf nicht zugreifen. Hier müssen wir leider große Einbußen hinsichtlich der Auswertbarkeit hinnehmen. Viele Ergebnisse und Ereignisse zur Laufzeit der Testfälle werden auf die Statuszeile ausgegeben bzw. gezeichnet und können mit unserem Testsystem nicht verglichen werden.

Jede Messagebox, jede Dialogbox und jedes Fenster sollte in seinem Titel von anderen unterscheidbar sein. Das Testsystem erkennt Ziele von Aktionen an ihrem Titelbezeichner. Sind nun beispielsweise gleichzeitig eine Dialogbox und eine Messagebox mit der Bezeichnung „Topographie“ offen, kann das Testsystem nicht wissen, auf welches der beiden Fenster die jeweilige Aktion ausgeführt werden soll. Aus diesem Grunde waren ein paar wenige „kosmetische“ Anpassungen im Quellcode des XCTL-Systems notwendig. So wurden die Messageboxen „Topographie“ und „Daten-Erhebung“ auf „Topographie - Information“ und „Daten-Erhebung - Information“ umbenannt, um sie von gleichnamigen Dialogboxen zu unterscheiden.

2.5 Grundlagen

2.5.1 Ressourcen-Dateien

Bei der Implementierung oberflächenbasierter Windows-Anwendungen werden neben den eigentlichen Programmfunktionalitäten auch ihre grafischen Benutzungsschnittstellen beschrieben. Während die Funktionalität im allgemeinen in den Quellcode-Dateien vorgenommen wird, findet die Beschreibung der Oberfläche hauptsächlich in den sogenannten RC-Dateien statt.

Damit die Testfälle relativ unabhängig gegenüber Änderungen an der Oberfläche bleiben, werden die aktuellen RC-Dateien der zu testenden Software als Grundlage zur Definition und Ausführung von Testsequenzen benutzt. Dazu wird eine Art Wissensbasis aus den Ressourcen zusammen gestellt, die alle Benutzerschnittstellen wie Dialogboxen, Menüs und Steuerelemente des Testobjektes in ihrem aktuellen Zustand enthält. Die Wartung und Aktualisierung dieser Datenbank unterliegt dem Tester bzw. den Entwicklern und ist maßgeblich für den Erfolg von Regressionstests verantwortlich. Im Abschnitt 3.4 wird der Umgang mit der Ressourcen-Datenbank beschrieben.

Der Aufbau von RC-Dateien ist in den verschiedenen Entwicklungsumgebungen (Borland, Microsoft) sehr ähnlich. Es werden sowohl optische als auch logische Informationen gespeichert. Optische Informationen beschreiben das Aussehen, die Position und Interaktionsmöglichkeiten. Logische Informationen beschreiben die Zugehörigkeit der Elemente zu einem Fenster bzw. zu einem Menü. Jedes Element wird innerhalb der Applikation durch einen eindeutigen Bezeichner (ID) identifiziert und angesprochen. Die genaue Syntax der Dateien hängt von der jeweiligen Entwicklungsumgebung ab. Hierzu ein Beispiel für eine RC-Datei in der Borland-Syntax:

```
ExecuteCmd DIALOG 38, 41, 165, 163 STYLE DS_MODALFRAME |
DS_NOIDLEMSG | WS_POPUP | WS_CAPTION | WS_SYSMENU CLASS "BorDlg"
CAPTION "Terminal" FONT 10, "Times New Roman" {
 CONTROL "Schalter", IDCANCEL, "BorBtn", BS_PUSHBUTTON | WS_CHILD |
    WS_VISIBLE | WS_TABSTOP, 124, 129, 33, 29
 EDITTEXT id_CommandLine, 8, 53, 57, 13, ES_LEFT | ES_UPPERCASE | WS_CHILD |
    WS_VISIBLE | WS_BORDER | WS_TABSTOP
 LTEXT " Kommando", -1, 17, 40, 41, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
 LTEXT " Antwort", -1, 74, 6, 38, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
 LISTBOX id_Response, 72, 18, 85, 106, LBS_NOTIFY | LBS_HASSTRINGS | LBS_DISABLENOSCROLL |
    WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL
 PUSHBUTTON "Text", cm_RotateMotor, 11, 135, 92, 17, WS_CHILD | WS_VISIBLE | WS_TABSTOP
 CONTROL "", 402, "BorShade", BSS_GROUP | BSS_LEFT | WS_CHILD | WS_VISIBLE, 5, 38, 64, 38
}
```

In diesem Auszug einer RC-Datei wird eine Dialogbox mit seinen zugehörigen Steuerelementen beschrieben. Allerdings sind nicht alle Informationen für das Testsystem bzw. für die Testfälle relevant. Optische Informationen spielen keine Rolle zur automatischen Ansteuerung der Elemente über Windows-Nachrichten (siehe Abschnitt 2.5.2), und können daher ignoriert werden. Lediglich die Informationen über Typ, Zugehörigkeit, Beschriftung und ID der Elemente sind zur Definition von Testfällen und zur automatisierten Fernsteuerung während ihrer Ausführung von Bedeutung.

Wie im Abschnitt 2.5.2 beschrieben, werden in den Windows-Nachrichten numerische IDs zur Identifizierung der Elemente benutzt. Die Zuordnung zwischen Ressourcen-Bezeichnern und ihren numerischen Pendants findet im allgemeinen in zusätzlichen Header-Dateien statt. Solche Dateien werden meistens von der Entwicklungsumgebung automatisch erzeugt. Im XCTL-System ist die Datei `rc_def.h` für die Zuordnung verantwortlich.

Elemente wie Menüs oder Dialogboxen, die erst zur Laufzeit des Programmes generiert werden, lassen sich nicht aus den RC-Dateien ablesen. Deshalb bietet der im Abschnitt 3.4 beschriebene *RCParser* die Möglichkeit, solche Elemente der Datenbasis manuell nachzutragen, um für die Erstellung bzw. Ausführung von Testfällen zur Verfügung zu stehen.

2.5.2 Die Windows-Nachrichtenverwaltung

Aus Sicht des Windows-Betriebssystems sind alle Oberfächenelemente Fenster. Sogar die Elemente eines Fensters, die sogenannten Controls, wie z.B. Buttons, Checkboxen, Eingabefelder usw., werden wie Fenster behandelt. Damit eine Funktionalität und eine Interaktion ermöglicht wird, ist es unabdingbar, dass die einzelnen Fenster miteinander oder auch das Betriebssystem selbst mit den Fenstern kommunizieren kann.

Die Grundlage für die Kommunikation bilden die Windows-Nachrichten. Jedes Fenster besitzt einen Nachrichtenpuffer, in der sich nacheinander die Nachrichten einordnen und abgearbeitet werden. Die Nachrichten können mehrere unterschiedliche Funktionen erfüllen. Zum Einen dienen sie als Anweisung an ein Fenster, dessen Zustand zu ändern oder eine bestimmte Aktion auszuführen. Hierunter fällt beispielsweise das Deaktivieren eines Steuerelementes (es wird grau dargestellt und ist vom Nutzer nicht mehr ansprechbar) oder das Annehmen eines Textes in einem Eingabefeld. Die zweite Art von Nachrichten dient zur Ermittlung des Zustandes oder bestimmter Daten. So läßt sich zum Beispiel herausfinden, ob eine Checkbox aktiviert ist oder nicht. Aber auch aktuelle Inhalte, wie Texte eines Eingabefeldes, können auf diese Weise ausgelesen werden. Die dritte Art von Nachrichten („Notifications“) dient dazu, andere Fenster über ein Ereignis zu informieren. Ein Button kann das Fenster, zu dem er gehört, davon informieren, dass er angeklickt worden ist. Dieses Ereignis kann dort abgefangen und behandelt werden.

Das folgende Beispiel zeigt API-Nachrichten, wie sie in MSVC++ implementiert werden müssten, um die ausgewählten Elemente einer Listbox abzufragen. Dazu wird zunächst die Anzahl der ausgewählten Elemente (`selcount`) der Listbox ermittelt, um mit einer weiteren Nachricht die eigentlichen Einträge in eine Liste (`selitems`) aufzunehmen. `IDC_LIST` ist dabei der eindeutige Identifizierer des Listen-Controls in der Dialogbox `hDlg`. `LB_GETSELCOUNT` und `LB_GETSELITEMS` sind die auszuführenden Aktionen auf diesem Control. Zusätzliche Parameter für einige Aktionen, wie zum Beispiel Speicherstellen für Rückgabewerte, finden im dritten und vierten Parameter (WPARAM, LPARAM) ihren Platz.

```
int  selcount = SendMessage( GetDlgItem(hDlg,IDC_LIST), \
                LB_GETSELCOUNT, 0, 0 );
int *selitems = new int[selcount];
SendMessage( GetDlgItem(hDlg,IDC_LIST), LB_GETSELITEMS, \
             (WPARAM)selcount, (LPARAM)(LPINT)selitems );
```

Während des Betriebs einer Oberflächenapplikation herrscht ein reger Nachrichtenaustausch. Die Nachrichten können sowohl zielgerichtet als auch an alle Fenster verschickt werden (Broadcast). Damit ein zielgerichtetes Verschicken möglich ist, besteht die Notwendigkeit, einzelne Fenster eindeutig zu identifizieren. Bei Steuerelementen ist dies relativ einfach. Sie erhalten, wie im Abschnitt 2.5.1 beschrieben, einen eindeutigen numerischen Wert, die sogenannte ID. Somit lässt sich leicht der Sender und der Empfänger einer Nachricht ermitteln. Fenster selbst lassen sich hingegen über mehrere Angaben identifizieren. Zum Einen besitzen sie mitunter die Prozess-ID ihres zugehörigen Prozesses. In unserem Testsystem wird allerdings zusätzlich auf die Identifikation über den Text in der Titelzeile zurückgegriffen, da die erstere Identifikationsmöglichkeit unzureichend bzw. aus technischen Gründen nicht realisierbar war.

Bei der Entwicklung einer Windows-Anwendung erfolgt die Zuordnung der Oberflächen-Elemente zu ihren IDs, wie im Abschnitt 2.5.1 beschrieben. Im Laufe der Studienarbeit wurde bereits eine Skriptsprache entwickelt, welche die Steuerung eines Programmes mit Windows(API)-Nachrichten über numerische IDs realisiert. Ein großer Nachteil dieser Sprach-Spezifikation besteht darin, dass sich während des Entwicklungsvorganges die Zuordnung zwischen Elementen und IDs ändern kann und somit ein Testskript für alte Programmversionen nicht mehr funktionsfähig wäre. Somit entstand die Idee, eine Skriptsprache zu entwickeln, welche, auch im Sinne der besseren Lesbarkeit, unabhängig von den numerischen IDs der Steuerelemente ist. Die Grundlage hierfür bildet die Identifikation der grafischen Oberflächenelemente über eindeutige Bezeichnungen. So bleiben dem Entwickler von Testfällen die Einzelheiten einer API-Nachricht verschlossen und er kann sich auf das Wesentliche konzentrieren. Beim Interpretieren der Testskripte werden die Bezeichnungen, mittels der in Abschnitt 2.5.1 angesprochenen Datenbasis, in numerische IDs für API-Nachrichten umgewandelt. Die Erstellung einer solchen Ressourcen-Datenbank wird mit dem *RCParser* realisiert (Abschnitt 3.4).

2.6 Aufgabenteilung

Diese Diplomarbeit erweitert die Ergebnisse unserer Studienarbeit [12] aus dem Wintersemester 2001/2002. Da die zu bearbeitende Problemstellung sehr umfangreich ist, war die Organisation und Koordinierung der Teilaufgaben von großer Wichtigkeit. Einen zentralen Teil dieser Arbeit stellt die Implementation des Testsystems als 32Bit-Windowsapplikation dar. Dabei handelt es sich um eine Menge von zusammen arbeitenden Programmen bzw. Komponenten.

Das Front-End der Test-Suite stellt die Schnittstelle zwischen dem Testobjekt und Testfällen her und bietet dem Tester die typischen Funktionen eines Testsystems. Es hat eine enge Beziehungen zum darunter liegenden Betriebsystem, kann Skripte interpretieren und GUI-Applikationen über Windows-Nachrichten fernsteuern, womit es softwaretechnisch zu einer sehr anspruchsvollen Aufgabe wird. Zur Verwaltung aller im Testobjekt vorliegenden Oberflächen-Elemente (Ressourcen), wird eine kleine Datenbank entwickelt, die vom Testsystem während der Entwicklung und Interpretation von Testskripten heran gezogen wird. Die Aufgaben zur Implementierung und Dokumentierung der genannten Komponenten, sowie der Entwurf einer Skriptsprache zur Fernsteuerung von GUI-Applikationen wurden hauptsächlich von Johann Letzel übernommen.

Folgende Aufgaben wurden überwiegend von Jens Hanisch bearbeitet: Entwurf der Teststrategie; Definition von Testfällen mit Ein- und Ausgabedaten, zu dem der Entwurf von Basis-Konfigurationsdateien gehört; Untersuchung der Vollständigkeit aller Testfälle (Methode der Instrumentierung); Implementierung einiger Teilkomponenten des Testsystems, wie Importfilter für attributierte Klassifikationsbäume und Dateivergleicher; Entwurf einer benutzerfreundlichen Skriptsprache zur Definition von Testskripten.

Teamarbeit hatte bei der Bewältigung dieser Arbeit oberste Priorität. Nicht zuletzt spiegelt sich diese Vorgehensweise im Design unseres Testsystems wieder, welches aus einer Sammlung wieder verwendbarer Komponenten besteht (siehe Kapitel 4). Monatliche Treffen und ein regelmäßiger Email-Austausch waren dabei wesentlich für die Verteilung und Bewältigung der Aufgaben.

2.7 Pflichtenheft *ATOS*

Pflichtenheft: ATOS v1.1

Johann Letzel (johann.letzel@johann-letzel.de)
Jens Hanisch (mail@jens-hanisch.de)

Version	Autor	Datum	Status	Kommentar
1.0	Johann Letzel Jens Hanisch	30.5.2002	Ersterfassung	
1.1	Johann Letzel Jens Hanisch	10.11.2002	Überarbeitung	

1. Zielbestimmung

Das Produkt soll die häufig anfallende Durchführung von Regressionstests oberflächenbasierter Windows-Anwendungen unterstützen. Im Vordergrund steht die Testautomatiserung einer Steuerungssoftware zur Halbleiter-Strukturanalyse.

1.1. Mußkriterien

Ziel ist die Entwicklung eines Testsystems mit ihren wesentlichen Funktionen:

- Anbieten einer Benutzungsoberfläche für den Tester zum Ausführen von Testaktionen (Zusammenstellen, Durchführen und Auswerten von Testfällen).
- Unterstütztung bei der Konstruktion von Testfällen.
- Auswertung der Resultate/Ereignisse des Testvorgangs.
- Verwaltung testrelevanter Daten (Testfälle, Testobjekt, testbegleitende Programme/Daten)

1.2. Wunschkriterien

- Verwaltung/Archivierung von Testprotokollen eines Testfalls.
- Erweiterung der Import-Schnittstelle zur automatischen Generierung von Testsequenzen.
- "Copy&Paste" von Aktionsschritten und Testsequenzen.

1.3. Abgrenzungskriterien

- Das Erstellen von attributierten Klassifikationsbäumen erfolgt über ein externes Werkzeug (*Classification Tree Editor*).
- Der Vergleich von Ausgabedateien mit Solldateien erfolgt über das entwickelte Werkzeug *DataDiff*.
- Der visuelle Vergleich von Grafikdateien mit Solldateien muss über einen externen Freeware-Grafikviewer erfolgen (z.B. *IrfanView*).
- Die Verwaltung der Projekt-Ressourcen (URF-Datenbasis) erfolgt über das entwickelte Werkzeug *RCParser*.

2. Produkteinsatz

Das Programm dient der Unterstützung eines Benutzers bei der Durchführung von automatisiert ablaufenden Regressionstests.

2.1. Anwendungsbereich

Regressionstests finden nach jeder Aktualisierung/Veränderung der Quellen des Testobjekts statt.

2.2. Zielgruppen

Entwickler und Tester der entwickelten Zielanwendung.

2.3. Betriebsbedingungen

Entwicklungs- und Einsatzumgebung.

3. Produktumgebung

Das Produkt läuft auf einem Arbeitsplatzrechner.

3.1. Software

Betriebsystem: Windows 9x/ME/NT/2000.

3.2. Hardware

PC mit den minimalen Anforderungen, die das installierte Betriebssystem benötigt.

3.3. Orgware

keine

3.4. Produktschnittstellen

- Testfälle können aus attributierten Klassifikationsbäumen durch eine Produktfunktion extrahiert und dem Testsystem zur Verfügung gestellt werden.

- Die Zuordnung zwischen in Skriptkommandos benutzten Bezeichnern für Oberflächen-Elemente des Testobjektes und ihren programminternen numerischen IDs wird über Dateien geregelt, die vom *RCParser* erstellt und verwaltet werden.

4. UseCase-Diagramm

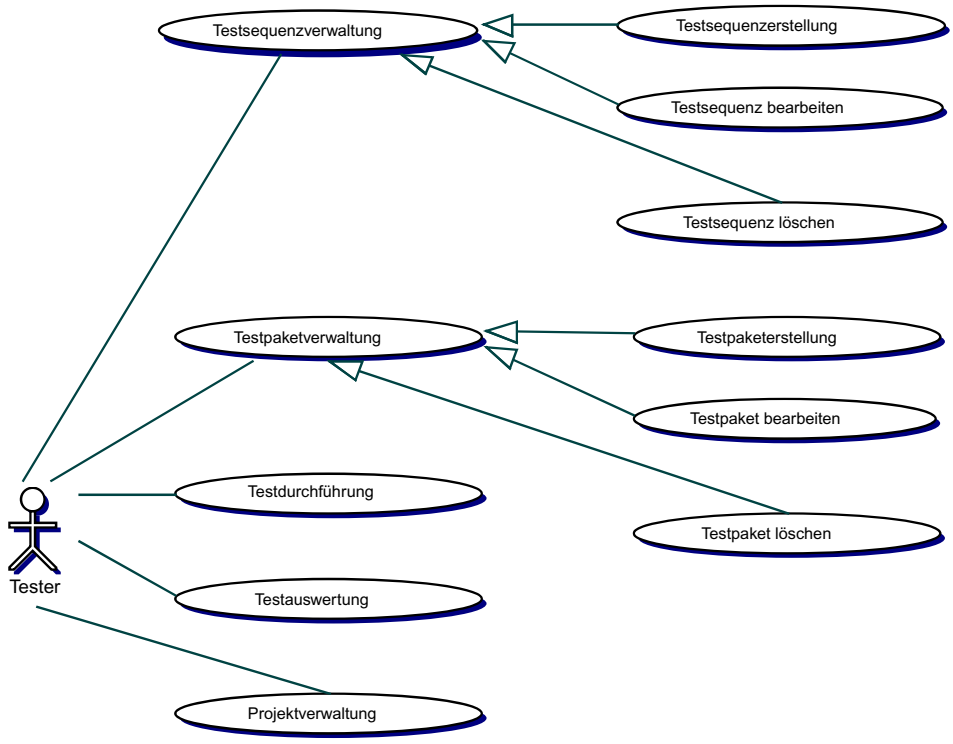


Abbildung 2.4: *ATOS* UseCase-Diagramm

5. Produktfunktionen

5.1. Testsequenzverwaltung

5.1.1. Testsequenzerstellung

/F10/ Anlegen von neuen Testsequenzen unter Vergabe eines Namens

/F20/ Import (CTE) mit Überprüfung und Fehlerbehandlung

/F21/ Auswahl der zu importierenden Datei

/F22/ Testsequenzauswahl und -übernahme

/F30/ Import einer Testsequenz direkt aus einer ASCII-Datei

5.1.2. Testsequenz bearbeiten

/F40/ Erstellung eines Testschritts mit Überprüfung

/F50/ Bearbeiten eines Testschritts mit Überprüfung

/F60/ Löschen eines Testschritts

5.1.3. Testsequenz löschen

/F70/ Löschen von Testsequenzen

5.2. Testpaketverwaltung

5.2.1. Testpaketerstellung

/F80/ Erstellung eines Testpakets unter Vergabe eines Namens

5.2.2. Testpaket bearbeiten

/F90/ Testsequenz(en) dem Paket hinzufügen

/F100/ Bearbeiten einer Testsequenz im Paket

/F110/ Testsequenz(en) im Paket anordnen und sortieren

/F120/ Testsequenz(en) aus dem Paket entfernen

5.2.3. Testpaket löschen

/F130/ Testpaket entfernen

5.3. Testdurchführung

/F140/ Auswahl einer Testsequenz oder eines Testpakets

/F150/ Testoptionen einstellen (Name des Testers, alle interaktiven Testsequenzen zuerst ausführen, ...)

/F160/ Testvorgang starten

/F170/ Testvorgang anhalten/wiederaufnehmen

/F180/ Testvorgang abbrechen

/F190/ Anzeigen des aktuellen Status' und des Aktionsfortschritts

5.4. Testauswertung

/F200/ Ausgabe von Fehlern (Ort, Grund)

/F210/ Testprotokoll/-auswertung

5.5. Projektverwaltung

/F220/ Testprojekt anlegen (Name, Testobjekt, Import von externen Dateien/Programmen)

/F230/ URF-Datenbasis (Kopie/Referenz) importieren

/F240/ Projekt bearbeiten

6. Produktdaten

/D10/ Testsequenzen (Gliederung in Aktionsschritte und Auswertungsschritte)

/D20/ URF-Datenbasis (Wissensbasis über alle Steuerelemente, Dialogboxen und Menüs, erzeugt aus den Quellen und Ressourcen des Testobjekts)

/D30/ Projektdaten (Testsequenzen, Testpakete, URF-Daten, Testobjekt, Name, externe Verweise)

/D40/ Testprotokolle/-auswertungen

7. Produktleistungen

/L10/ Latenzzeiten beim Ausführen von Aktions- bzw. Auswertungsschritten sind möglichst gering zu halten.

8. Benutzungsschnittstelle

/B10/ Die Bedienung soll menüorientiert sein.

/B20/ Standardmäßig ist eine Mausbedienung vorgesehen.

/B30/ Die Oberfläche soll auf Mausbedienung ausgelegt sein. Aber auch eine Bedienung per Tastatur soll möglich sein.

9. Globale Teställe

/T10/ Import von Testsequenzen

/T20/ Testdurchführng

/T30/ Projektkonsistenz

/T40/ Konsistenzfehler von Testsequenz mit URF-Datenbasis

10. Entwicklungsumgebung

- Betriebssystem: Windows 2000
- Entwicklungsumgebung: Microsoft Visual C++ 6

11. Qualitätsbestimmung

Produktqualität	**Sehr gut**	**Gut**	**Normal**	**Nicht relevant**
Funktionalität				
Angemessenheit		√		
Richtigkeit		√		
Interoperabilität		√		
Ordnungsmäßigkeit		√		
Sicherheit		√		
Zuverlässigkeit				
Reife			√	
Fehlertoleranz		√		
Wiederherstellbarkeit			√	
Benutzbarkeit				
Verständlichkeit	√			
Erlernbarkeit	√			
Bedienbarkeit	√			
Effiziens				
Zeitverhalten		√		
Verbrauchsverhalten			√	
Änderbarkeit				
Analysierbarkeit			√	
Modifizierbarkeit		√		
Stabilität			√	
Prüfbarkeit			√	
Übertragbarkeit				
Anpaßbarkeit		√		
Installierbarkeit		√		
Austauschbarkeit		√		

2.8 Pflichtenheft *RCParser*

Pflichtenheft: RCParser v1.0

Johann Letzel (johann.letzel@johann-letzel.de)

Version	Autor	Datum	Status	Kommentar
1.0	Johann Letzel	30.05.2002	Ersterfassung	

1. Zielbestimmung

Das Produkt soll Informationen über die Oberfläche einer *Windows*^{TM}-Anwendung sammeln und die Möglichkeit bieten diese zu bearbeiten um die extrahierten Daten des Testsystems *ATOS* zur Verfügung zu stellen.

1.1. Mußkriterien

Ziel ist die Entwicklung eines oberflächenbasierten Werkzeugs mit den wesentlichen Funktionen:

- Extrahieren von Oberflächeninformationen aus *Ressourcen*-Dateien und *Header*-Dateien, wobei Flexibilität gegenüber syntaktischen Variationen ermöglicht werden soll.
- Anbieten der Möglichkeit zur Bearbeitung der gesammelten Informationen.
- Unterstützung bei der Lösung von Konflikten innerhalb der Informationen.
- Exportieren der Informationen in eine *URF*-Datei für das Testsystem *ATOS*.
- Verwaltung der verwendeten Dateien in einem Projekt.

1.2. Wunschkriterien

- Hinreichend schnelle Aktualisierung des Datenbestands nach Veränderung der verwendeten Quelldateien.

1.3. Abgrenzungskriterien

- Die *Ressourcen-* und *Header*-Dateien werden von verschiedenen Entwicklungsumgebungen erzeugt, z.B. *Microsoft Visual C++* oder *Borland C++*.

2. Produkteinsatz

Das Programm dient zur Unterstützung eines Benutzers bei der Extraktion von Oberflächeninformationen der zu testenden Anwendung für das Testsystem *ATOS*.

2.1. Anwendungsbereich

Die Aktualisierung der Informationen findet nach jeder Veränderung der Oberfläche des Testobjekts statt.

2.2. Zielgruppen

Entwickler und Tester der entwickelten Zielanwendung.

2.3. Betriebsbedingungen

Entwicklungs- und Einsatzumgebung.

3. Produktumgebung

Das Produkt läuft auf einem Arbeitsplatzrechner.

3.1. Software

Betriebsystem: Windows 9x/ME/NT/2000.

3.2. Hardware

PC mit den minimalen Anforderungen, die das installierte Betriebssystem benötigt.

3.3. Orgware

keine

3.4. Produktschnittstellen

- Die *Ressourcen*- und *Header*-Dateien werden von der verwendeten Entwicklungsumgebung zur Verfügung gestellt.

4. UseCase-Diagramm

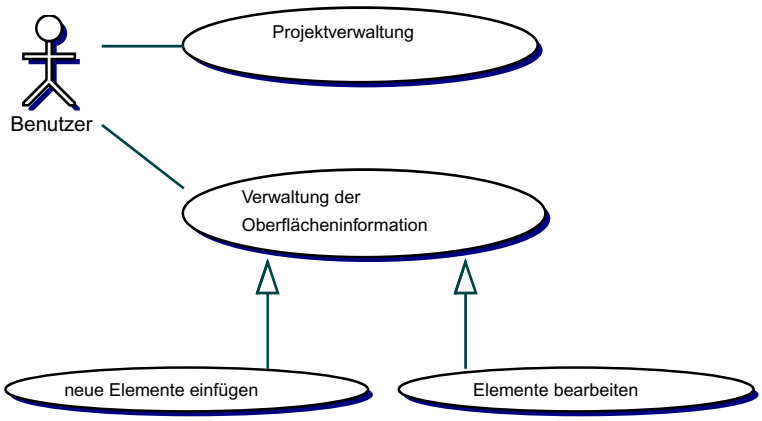


Abbildung 2.5: *RCParser* UseCase-Diagramm

5. Produktfunktionen

5.1. Projektverwaltung

/F10/ Anlegen eines neuen Projekts.

/F20/ Importieren von *Ressourcen*-Dateien.

/F21/ Auswahl der zu importierenden Datei(en).

/F22/ Auswahl eines passenden Importfilters für das zu importierende Format.

/F30/ *Ressourcen*-Dateien aus dem Projekt entfernen.

/F40/ Importieren von *Header*-Dateien.

/F41/ Auswahl der zu importierenden Datei(en).

/F50/ *Header*-Dateien aus dem Projekt entfernen.

/F60/ Projekt nach Änderungen an den *Header*- oder *Ressourcen*-Dateien aktualisieren.

/F70/ Gesammelte Oberflächeninformationen in eine URF-Datei exportieren.

5.2. Verwaltung der Oberflächeninformationen

5.2.1. Elemente einfügen

/F80/ Direkt neue Elemente einfügen (Dialoge, Steuerelemente, Menüs, Untermenüs und Menüeinträge).

/F90/ Elemente aus einer Vorlage erzeugen und einfügen (Dialoge).

/F100/ Importieren von Elementen. (Wird mittels der Funktionen **/F20/** bis **/F22/** und **/F40/** bis **/F41/** realisiert).

5.2.2. Elemente bearbeiten

/F110/ Elemente umbenennen (Dialoge, Steuerelemente, Untermenüs und Menüeinträge).

/F120/ Eigenschaften von Elementen bearbeiten.

/F130/ Konflikte von Elementen anzeigen.

/F140/ Konflikte von Elementen lösen (lösbare Konflikte werden duch **/F110/** oder **/F120/** ermöglicht).

/F150/ Elemente entfernen bzw. in den Originalzustand zurücksetzen.

/F160/ Elemente de-/aktivieren.

6. Produktdaten

/D10/ Verwendete *Ressourcen*-Dateien (Herkunft, Name und Format).

/D20/ Verwendete *Header*-Dateien (Herkunft, Name).

/D30/ Projektdaten (*Ressourcen*-, *Header*-Dateien, akt. Stand der Oberflächeninformationen)

7. Produktleistungen

/L10/ Latenzzeiten beim Ausführen von Bearbeitungs- und Aktualisierungsschritten sind möglichst gering zu halten.

8. Benutzungsschnittstelle

/B10/ Die Bedinung soll menüorientiert sein.

/B20/ Standardmäßig ist eine Mausbedienung vorgesehen.

/B30/ Die Oberfläche soll auf Mausbedienung ausgelegt sein. Aber auch eine Bedienung per Tastatur soll möglich sein.

9. Globale Teställe

/T10/ Import von *Ressourcen*- und *Header*-Dateien

/T20/ Bearbeitung der Oberflächeninformationen

/T30/ Export der gesammelten Informationen in eine URF-Datei.

10. Entwicklungsumgebung

- Betriebssystem: Windows 2000
- Entwicklungsumgebung: Microsoft Visual C++ 6

11. Qualitätsbestimmung

Produktqualität	**Sehr gut**	**Gut**	**Normal**	**Nicht relevant**
Funktionalität				
Angemessenheit		√		
Richtigkeit		√		
Interoperabilität		√		
Ordnungsmäßigkeit		√		
Sicherheit		√		
Zuverlässigkeit				
Reife			√	
Fehlertoleranz		√		
Wiederherstellbarkeit			√	
Benutzbarkeit				
Verständlichkeit	√			
Erlernbarkeit	√			
Bedienbarkeit	√			
Effiziens				
Zeitverhalten		√		
Verbrauchsverhalten			√	
Änderbarkeit				
Analysierbarkeit			√	
Modifizierbarkeit		√		
Stabilität			√	
Prüfbarkeit			√	
Übertragbarkeit				
Anpaßbarkeit		√		
Installierbarkeit		√		
Austauschbarkeit		√		

2.9 Pflichtenheft *DataDiff*

Pflichtenheft: DataDiff v1.0

Jens Hanisch (mail@jens-hanisch.de)

Version	Autor	Datum	Status	Kommentar
1.0	Jens Hanisch	30.5.2002	Ersterfassung	

1. Zielbestimmung

Das Produkt soll einen Vergleich beliebiger Textdateien mit Referenzdateien ermöglichen.

1.1. Mußkriterien

Ziel ist die Entwicklung eines Dateivergleichers mit folgenden Funktionen:

- Interpretation von Vergleichskommandos (Metainformationen) in den Referenzdateien zur Steuerung der Leseposition und Behandlung der eingelesenen Zeichenketten (numerischer vs. textueller Vergleich).
- Ausschreibung der Vergleichs-Resultate in einer Logdatei.

1.2. Wunschkriterien

keine

1.3. Abgrenzungskriterien

keine

2. Produkteinsatz

Das Programm dient zum Vergleich von Ausgabedateien einer Testapplikation mit Erwartungswerten im Rahmen von Regressionstestläufen.

2.1. Anwendungsbereich

Regressionstestläufe mit einhergehenden Gegenüberstellungen von Ausgabedateien und Dateien aus früheren Versionen, finden nach jeder Aktualisierung/Veränderung der Quellen des Testobjektes statt.

2.2. Zielgruppen

Entwickler und Tester der entwickelten Zielanwendung.

2.3. Betriebsbedingungen

Entwicklungs- und Einsatzumgebung.

3. Produktumgebung

Das Produkt läuft auf einem Arbeitsplatzrechner.

3.1. Software

Betriebsystem: Windows 9x/ME/NT/2000.

3.2. Hardware

PC mit den minimalen Anforderungen, die das installierte Betriebssystem benötigt.

3.3. Orgware

keine

3.4. Produktschnittstellen

keine

4. Produktfunktionen

4.1. Positionierung der Leseposition

/F10/ Überspringen aller Zeichen bis zum Ende der aktuellen Zeile

/F20/ Zeichenkette darf an beliebiger Stelle in der Textdatei vorkommen; Leseposition kann wieder an die aktuelle Stelle zurück gesetzt werden (optional)

4.2. Vergleich von eingelesenen Zeichenketten

4.2.1. Numerischer Vergleich

/F30/ Istwert gleich Sollwert

/F40/ Istwert kleiner Sollwert

/F50/ Istwert kleiner-gleich Sollwert

/F60/ Istwert größer Sollwert

/F70/ Istwert größer-gleich Sollwert

/F80/ Differenz von Istwert und Sollwert liegt in einem Toleranzbereich

/F90/ Zyklischer Vergleich von Istwerten mit Sollwerten unter Beachtung von Toleranzen (geeignet für Datenstrukturen im Spaltenformat)

4.2.2. Textueller Vergleich

/F100/ Jede eingelesene Zeichenkette wird zeichenweise mit der Zeichenkette aus der Referenzdatei verglichen

4.3. Protokollierung

/F110/ Das Ergebnis der Durchführung jedes Vergleiches, ob textuell oder numerisch, wird in eine Logdatei ausgeschrieben

/F120/ Die gleichzeitige Ausgabe auf dem Bildschirm kann optional abgeschaltet werden, um den Vergleichsprozess zu beschleunigen

5. Produktdaten

/D10/ Referenzdateien bestehend aus zu vergleichenden Zeichenketten und numerischen Werten sowie Kommandos zur Behandlung der Zeichenketten und Positionierung der Leseposition währen des Vergleichsvorganges

6. Produktleistungen

/L10/ Vergleich von Ausgabe- und Referenzdateien soll möglichst zügig verlaufen, um den gesamten Prozess des Regressionstests zu beschleunigen.

7. Benutzungsschnittstelle

/B10/ Die Bedienung erfolgt ausschließlich über Parameter beim Aufruf über einen MS-DOS-Kommandointerpreter.

8. Globale Teställe

/T10/ Vergleich einer Datei mit sich selbst als Referenzdatei darf keine Fehler hervorufen

/T20/ Vergleich einer Datei mit textuellen Abweichungen

/T30/ Vergleich einer Datei mit numerischen Abweichungen

9. Entwicklungsumgebung

- Betriebssystem: Windows 2000

- Entwicklungsumgebung: Microsoft Visual C++ 6

10. Qualitätsbestimmung

Produktqualität	**Sehr gut**	**Gut**	**Normal**	**Nicht relevant**
Funktionalität				
Angemessenheit		√		
Richtigkeit		√		
Interoperabilität		√		
Ordnungsmäßigkeit		√		
Sicherheit		√		
Zuverlässigkeit				
Reife			√	
Fehlertoleranz		√		
Wiederherstellbarkeit			√	
Benutzbarkeit				
Verständlichkeit		√		
Erlernbarkeit		√		
Bedienbarkeit			√	
Effiziens				
Zeitverhalten	√			
Verbrauchsverhalten		√		
Änderbarkeit				
Analysierbarkeit			√	
Modifizierbarkeit			√	
Stabilität			√	
Prüfbarkeit			√	
Übertragbarkeit				
Anpaßbarkeit			√	
Installierbarkeit			√	
Austauschbarkeit			√	

Kapitel 3

Das Testsystem

3.1 Allgemeine Arbeitsweise

Das Zusammenspiel der Komponenten des Testsystems wird schematisch in Abbildung 3.1 dargestellt. Mittelpunkt bildet die Testsuite *ATOS* (Automatisiertes Testen oberflächenbasierter Systeme). Von dort aus wird der gesamte Testprozess durchgeführt. Das Testobjekt wird durch entsprechende Skriptkommandos von *ATOS* als neuer Prozess gestartet und angesteuert. Das XCTL-Steuerprogramm als Testobjekt seinerseits arbeitet wie gewohnt, lädt dazu seine notwendigen Konfigurationsdateien und erstellt während seiner Arbeit gegebenenfalls Ausgabedateien zur Datenaufnahme oder zur Protokollierung.

Die Konfigurationsdateien werden als Eingabedaten des Testobjektes behandelt und müssen bei der Testdurchführung dem XCTL-System zur Verfügung gestellt werden. Vor dem Start des XCTL-Systems werden dazu die Konfigurationsdateien `HARDWARE.INI` und `DEVELOP.INI` in Abhängigkeit der jeweiligen Testsequenz durch präparierte Dateien, wie `\env\OTOPO_HARDWARE.INI` und `\env\TEST_DEVELOP.INI` vom Testsystem temporär ersetzt. Nach Beendigung oder bei Abbruch des Regressionstestes wird der ursprüngliche Zustand mit den originalen Konfigurationsdateien wieder hergestellt.

Zur Ausführung von Testschritten, die am Testobjekt durchgeführt werden sollen, interpretiert *ATOS* Kommandos aus Dateien in der Skriptpsrache HTS (siehe Abschnitt 3.5.3).

Die Ausführung jedes Testschrittes wird dabei in eine Protokolldatei geschrieben, so dass bei Problemen, die während eines Testdurchlaufs entstanden, nach dessen Ursache gesucht werden kann. Dazu kann die Fehlermeldung des zuletzt gescheiterten Skriptkommandos sehr hilfreich sein.

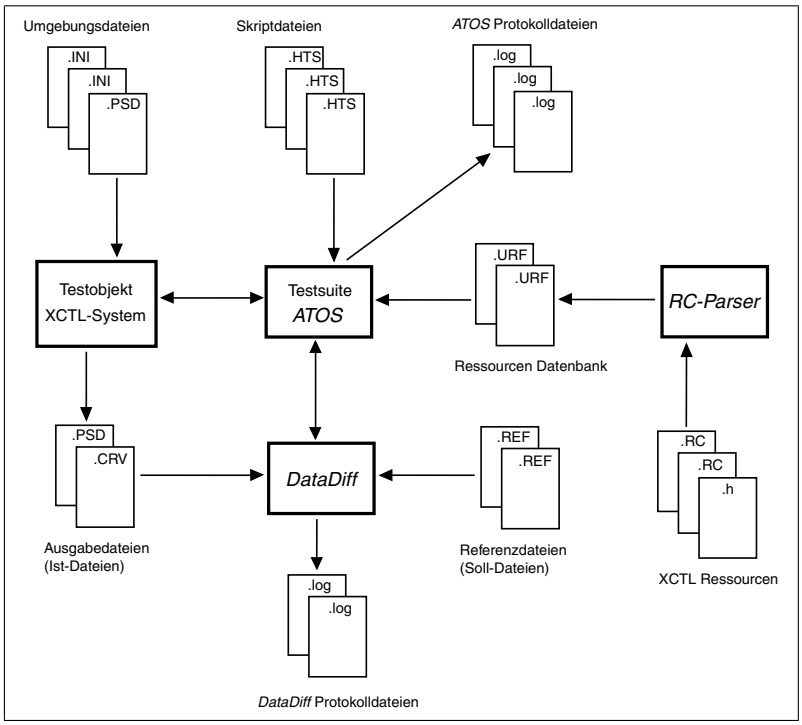


Abbildung 3.1: Beziehungen der Testkomponenten

Das XCTL-System könnte zum Beispiel nicht mehr auf Eingaben reagieren, weshalb die Ausführung des aktuellen Skriptes abbricht, wenn sich das erwartete Ergebnis nicht einstellt. Aber auch Aktionsschritte können scheitern und zum Abbruch führen. So bricht die Durchführung ab, wenn ein Button in einer Dialogbox gedrückt werden soll, welcher gar nicht mehr existiert, weil die Dialogbox unerwartet vorzeitig geschlossen wurde.

Essentielle Grundlage für die Interpretation der Skriptkommandos ist die Datenbasis über alle verfügbaren Steuerelemente, Dialogboxen, Fenster und Menüs des Testobjektes. Diese Datenbank wird mit dem *RCParser* erstellt und verwaltet und liegt *ATOS* zur Laufzeit der Testskripte vor. (siehe dazu Abschnitt 3.4)

Werden nach Ausführung einer Testsequenz Ausgabedateien vom XCTL-System erzeugt, so kann *ATOS* durch entsprechende Skriptkommandos das Dateivergleichsprogramm *DataDiff* aufrufen. Dieser 32-Bit Konsolenanwendung werden die Namen der erzeugten Ausgabedatei (Istdatei), sowie einer Referenzdatei (Solldatei) als Parameter übergeben. Auf die Konstruktion von Referenzdateien wird im Abschnitt 3.7 genauer eingegangen.

Während des Vergleichs der Daten wird das Protokoll `DataDiff.log` erstellt. War der Dateivergleich erfolgreich, wird die Ausführung des Skriptes fortgesetzt. Bei Unstimmigkeiten zwischen den Ist- und Solldaten wird das aktuelle Testskript mit einer Fehlermeldung abgebrochen, der Ausgangszustand wieder hergestellt und die Ausführung mit dem nächsten Skript aus der Auswahlliste fortgesetzt.

3.2 Durchführung von Regressionstests

Für die Verwaltung getesteter Versionen des XCTL-Systems ist die Quelltext-Versionskontrolle CVS sehr hilfreich. Sie ermöglicht das Einfügen von sogenannten „Tags“, welche alle Quellen aus dem CVS-Modul in ihrem aktuellsten Zustand kennzeichnen, um im Laufe der Projektarbeit auf diese zugreifen zu können. Auf diese Weise kann eine bestimmte Version im CVS-Repository festgehalten werden. Lauffähige und vollständig getestete Versionen werden mit einer eindeutigen Marke (Tag), die das Datum des Testdurchlaufs beinhaltet, versehen.

Ein Regressionstest wird immer auf der aktuellsten Version des Testobjekts durchgeführt. Vom Tester muss sichergestellt werden, dass auch wirklich die überprüfte Version im CVS-Repository markiert wird und nicht eine Version, die nach der Testdurchführung verändert wurde. Eine Möglichkeit dazu ist das „Einfrieren“ einer Version mittels einer Marke direkt vor dem Test. War der Test erfolgreich, darf die Marke im CVS-Repository liegen bleiben, bei einem Misserfolg muss sie dagegen wieder entfernt werden.

Ein vollständiger Regressionstests wird mit den folgenden Schritten durchgeführt:

1. `% cvs co XC010701`
   Aktuelle Testobjektversion aus dem CVS-Repository holen.
   Dabei wird ein Verzeichnis mit dem Namen des Moduls im Homeverzeichnis des Testers erstellt. In diesem Kommando ist der Name des Moduls: `XC010701`.

2. `% cvs tag Test<Datum> XC010701`
   Markiert alle aktuellen Quelldateien im CVS-Repository, die sich im selben Zustand wie die Dateien im Homeverzeichnis des Testers befinden. `<Datum>` ist durch das aktuelle Datum des Regressionstestes zu ersetzen.

3. Kopieren des Verzeichnisses aus dem Homeverzeichnis in ein temporäres Arbeitsverzeichnis, in dem der Test durchgeführt werden kann. Das Zielverzeichnis darf auf einem beliebigen PC mit Windows9x oder WindowsNT liegen.

4. Kompilieren der Quellen und Erstellen einer ausführbaren Programmversion im erstellten Arbeitsverzeichnis.

5. **Installation** des Testsystems *ATOS*, wie in Abschnitt 3.2.1 beschrieben.

6. Durchführung des **Regressionstests**, wie in Abschnitt 3.2.2 beschrieben.

7. **Auswertung** des Regressionstestlaufs, wie in Abschnitt 3.2.3 beschrieben.

8. Nach einem erfolgreichen Testdurchlauf:

   i Die Projektteilnehmer sind davon zu informieren, dass die Version mit der Marke `Test<Datum>` im CVS-Repository den Regressionstest bestanden hat.

   Nach einem gescheiterten Testdurchlauf:

   i Auf das Homeverzeichnis des Testers, in dem das Verzeichnis mit dem Namen des Moduls liegt, zurück wechseln.

   ii `% cvs tag -d Test<Datum> XC010701`
   Die aktuelle Version im CVS darf nicht markiert bleiben, da sie sich in einem fehlerhaften Zustand befindet!

   iii Die Projektteilnehmer sind davon zu informieren, dass die zuletzt aktualisierte Version im CVS-Repository den Regressionstest nicht bestanden hat. Die gescheiterten Testsequenzen sind mit Hilfe der Protokollausgaben zusammenzustellen, um eine gezielte Fehlerverfolgung seitens der Entwickler starten zu können.

Vor der Durchführung eines Testdurchlaufs sollte der Tester die anderen Projektteilnehmer darauf hinweisen, an den aktuellen Quellen keine weiteren Veränderungen vorzunehmen, bis die Arbeit des Regressionstestes beendet ist. Schlägt der Testdurchlauf fehl, müssen zunächst die dafür verantwortlichen Veränderungen ausfindig gemacht und repariert werden. Erst danach dürfen Aktualisierungen an den Quellen vorgenommen werden. Somit kann verhindert werden, dass Entwickler ihre fehlerhaften Codezeilen, die zum Scheitern des Regressionstestes führen, in der Zwischenzeit erweitern und erneut in das CVS-Repository einspielen.

Die Quelltext-Versionskontrolle CVS bringt noch einige nützliche Kommandos mit sich, die bei der Arbeit mit dem Tagging-Mechanismus sinnvoll sein können.

```
% cvs co -r Test200701 XC010701
```

Erstellt eine gültige und getestete Version aus dem CVS-Repository, die mit der Marke `Test200701` versehen wurde. Das erstellte Verzeichnis mit den Quelldateien steht immer noch unter der Kontrolle von CVS und kann somit als Ausgangspunkt für darauf aufbauende Versionen benutzt werden.

```
% cvs export -r Test200701 XC010701
```

Die Exportfunktion hingegen erstellt ein Verzeichnis mit den Quellen aus dem CVS-Repository, das nicht mehr unter der weiteren Kontrolle von CVS steht.

```
% cvs status -v XC010701
```

Um sich die Tags zu einem Modul anzeigen zu lassen, ist die Option `-v` sehr hilfreich. Dieses Kommando gibt zum Beispiel alle Tags des Moduls `XC010701` aus.

Da die Marke eines erfolgreichen Testdurchlaufs sehr wichtig für die - Regressionstest unterstützte - Entwicklung ist, wird sie in den Testpaketen zu den jeweiligen Anwendungsfällen im Web-Repository mitgeführt. Dabei kennzeichnet die Marke im Kopf des Dokumentes die zuletzt markierte Version im CVS-Repository, welche einen erfolgreichen Durchlauf mit allen zugehörigen Testfällen dieses Testpaketes absolvieren konnte.

3.2.1 Installation

Alle benötigten Komponenten für einen automatisierten Regressionstest können vom selben Server bezogen werden, auf dem auch die Quelltextverwaltung des XCTL-Systems stattfindet. Zur Installation genügt somit die gleiche Vorgehensweise, die auch für die Besorgung einer aktuellen Version des XCTL-Systems aus dem CVS-Repository notwendig ist. Nach dem Einloggen auf einen UNIX- bzw. Linux-Rechner im Informatik-Institut der Humboldt-Universität, zum Beispiel über den Dienst „Telnet“, muss zunächst die Umgebungsvariable `CVSROOT` auf das CVS-Repository gesetzt werden:

```
% setenv CVSROOT /vol/baal-vol3/projekt98/quellen
```

Ein einfaches Kommando zum Auschecken genügt, um ein Verzeichnis mit den Komponenten des Testsystems im Homeverzeichnis zu erstellen:

```
% cvs co ATOS
```

Das Projektverzeichnis für *ATOS* mit allen Testskripten, Umgebungsdateien und Referenzdateien besorgt man sich auf die gleiche Weise.

```
% cvs co ATOS_XCTLProjekt
```

Die erstellten Verzeichnisse `.\ATOS` und `.\ATOS_XCTLProjekt` einschließlich aller Unterverzeichnisse sind dann nur noch auf den jeweiligen Entwickler-PC, in ein beliebiges Arbeitsverzeichnis zu kopieren. Selbstverständlich muss die aktuelle und zu testende XCTL-Version in kompilerter und ausführbarer Form ebenfalls in einem Verzeichnis auf diesem Entwickler-PC vorliegen. Die Zugriffsrechte müssen gewährleisten, dass das Programmverzeichnis des XCTL-Systems für das Testsystem erreichbar ist. Die Tabelle 3.1 beschreibt die Unterverzeichnisse von `.\ATOS_XCTLProjekt`, auf die *ATOS* während eines Regressionstests zugreift.

Werden für die Durchführung von Testsequenzen zusätzliche ausführbare Programme benötigt, finden sie im Verzeichnis `\bin\` ihren Platz. Für einen augenscheinlichen Vergleich von Meßkurven wird der Freeware-Grafikviewer *IrfanView* von Irfan Skiljan eingesetzt [25]. Er ist für nicht kommerzielle Zwecke kostenlos, unterstützt mit einer Größe von nur 424 KByte eine Vielzahl von Grafikformaten und lässt sich mit einer INI-Datei sehr leicht konfigurieren.

Verzeichnisname	Inhalt
.\bin\	testbegleitende Programme (z.B. DataDiff.exe, i_view32.exe)
.\ref\	Referenzdateien und Kurvengrafiken
.\env\	präparierte Umgebungsdateien für die Testfälle (z.B. INI-Dateien)
.\seq\	Testskripte (Endung *.HTS)
.\log\	Logdateien für jedes ausgeführte Skript/Testpaket
.\urf\	„Uniform Resource File“ (Datenbank(en) über alle Steuerelemente, Fenster und Menüs des Testobjekts)
.\cte\	attributierte CTE-Diagramme zur automatisierten Gewinnung von Testsequenzen

Tabelle 3.1: Verzeichnisstruktur eines ATOS-Projektes

Das Verzeichnis \ref\ beinhaltet alle Solldateien, die für einen Dateivergleich mit *DataDiff* oder für einen Kurvenvergleich mit *IrfanView* während der Testdurchführung herangezogen werden. Im Verzeichnis \env\ liegen die aktuellen und gültigen Konfigurationsdateien, wie TEST_DEVELOP.INI, OTOPO_HARDWARE.INI oder 1DIFF_HARDWARE.INI. Als Eingabedaten für das XCTL-System stehen diese Dateien in enger Beziehung zu den Testsequenzen. Eine Datenbank über alle Steuerelemente, Fenster, Dialogboxen und Menüs, die im Testobjekt angesprochen werden können, wird im Verzeichnis \urf\ gehalten. Dabei können zu einem Testobjekt sogar verschiedene Datenbanken geführt werden. Näheres dazu ist im Abschnitt 3.4 nachzulesen.

Da sich die Struktur und die Inhalte dieser Dateien durch die ständige Überarbeitung der Komponenten des XCTL-Systems verändern können, müssen immer die aktuellsten Dateien aus dem Modul ATOS_XCTLProjekt des CVS-Repository's für einen Regressionstest vorliegen. Dabei ist die Versionsverwaltung von CVS sehr hilfreich. Durch den beschrieben Installationsvorgang (cvs co ...) erhält man immer den aktuellsten Stand der Referenz-, Umgebungs- und Ressourcendateien.

Das Verzeichnis \seq\ beinhaltet alle aktuellen Testskripte, die bei der Zusammenstellung von Testpaketen zur Verfügung stehen. Die Endung .HTS macht Dateien als Skriptkommando-Dateien erkenntlich. Prinzipiell können Skriptdateien für *ATOS* mit beliebigen Endungen versehen werden. Der Dateiname sollte eine eindeutige Skript-ID enthalten, welche mit der Testsequenz-ID im Web-Repository korrespondiert.

Das Verzeichnis \log\ ist nach der Installation leer. Hier werden Protokolle währen der Durchführung von Testsequenzen angelegt. Auch Ausgabeprotokolle externer Programme wie *DataDiff* liegen in diesem Verzeichnis zur Einsicht bereit.

Im Verzeichnis \cte\ liegen attributierte CTE-Diagramme, die sich zur automatisierten Generierung von Testskripten eigenen. Der Umgang mit attributierten Klassifikationsbäumen wird ausführlich im Abschnitt 3.8.1 beschrieben.

Die Verzeichnisstruktur von .\ATOS wird in Tabelle 3.2 beschrieben. Neben dem ausführbaren Programm ATOS.exe sind wichtige Regeldateien zur Verarbeitung von CTE-Diagrammen und HTS-Skripten (cte.rul, hts2ats.rul, hts.syn) im Hauptverzeichnis vorzufinden.

Verzeichnisname	**Inhalt**
.\RCParser\	Verwaltungsprogramm für Datenbasis über alle Steuerelemente, Fenster und Menüs des Testobjekts
.\src\	MSVC++ Quelltexte von *DataDiff*, *ATOS* und *RCParser*

Tabelle 3.2: Verzeichnisstruktur des Testsystems

Im Verzeichnis \RCParser\ befindet sich das ausführbare Programm RCParser.exe zur Verwaltung aller Steuerelemente, Fenster, Dialogboxen und Menüs des Testobjektes, sowie Konfigurations- und Regeldateien zum Verarbeiten der zugrunde liegenden Ressourcen. Zur Einsicht und eventuellen Bearbeitung der Quellen von *ATOS*, *RCParser* und *DataDiff* steht das Verzeichnis \src\ zur Verfügung.

3.2.2 Regressionstest

Liegen alle notwendigen Konfigurationsdateien, Testskripte und Referenzdateien in den Verzeichnissen \env\, \seq\ und \ref\ vor, kann ein Regressionstest durchgeführt werden. Gestartet wird der automatisierte Testvorgang durch Ausführen der Datei ATOS.exe aus dem Rootverzeichnis .\ATOS. Nach Einlesen oder Erstellen einer Projektdatei bietet sich dem Benutzer eine komfortable Testumgebung (Abbildung 3.2). Detailierte Anweisungen zum Umgang mit *ATOS* sind im Benutzerleitfaden [19] zu finden.

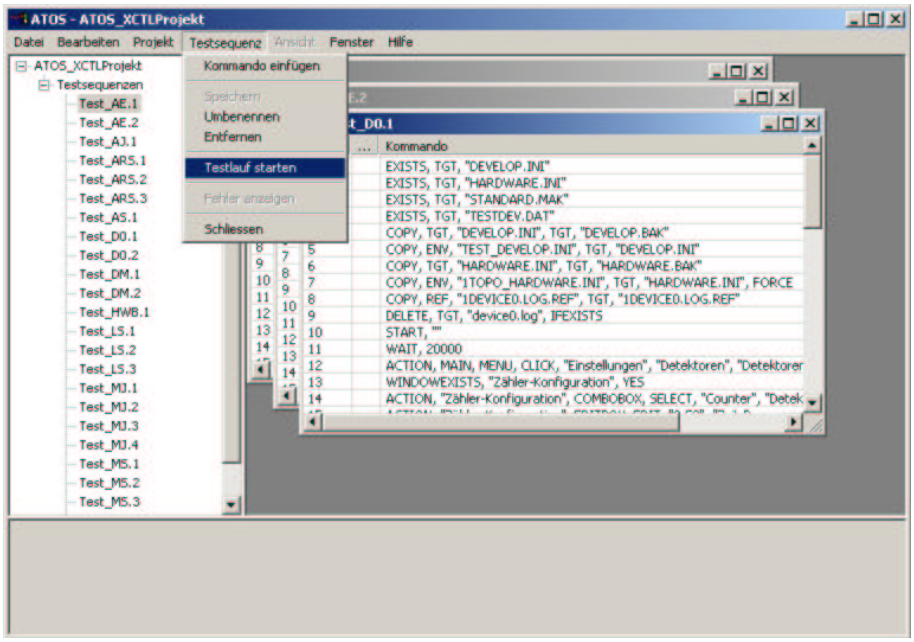


Abbildung 3.2: Die Testsuite *ATOS*

Für einen Regressionstest sind alle gewünschten Testpakete bzw. Testskripte auszuwählen und durchzuführen. Ein vollständiger Regressionstest erzwingt die Auswahl aller Testskripte, die sich im Verzeichnis \seq\ befinden. Nur wenn alle Testskripte fehlerfrei durchlaufen, kann sicher gestellt werden, dass die Korrektheit aller Funktionalitäten aus früheren Versionen überprüft wurden.

Visuelle Kurvengrafik-Vergleiche und einige Arbeiten mit der Maus müssen während eines Testdurchlaufs vom Tester interaktiv durchgeführt werden. Das Testsystem ist an diesen Stellen auf die Reaktion des Testers angewiesen und führt solange keine weiteren Skriptkommandos aus. Anweisungen dazu erhält der Tester in Form von Messageboxen. Skripte, die in ihrer Ausführung den Eingriff des Testers erwarten, sollten an den Anfang der Auswahlliste gestellt werden, um den Vorgang des Testdurchlaufs nicht von Anfang bis zum Ende überwachen zu müssen. Dazu kann *ATOS* selbständig erkennen, ob bestimmte Skripte interaktive Eingriffe erwarten und sie entsprechend in die Ausführungsliste einsortieren.

Das Testsystem beendet seine Arbeit entweder beim expliziten Abbruch durch den Tester oder nach erfolgreicher Ausführung des letzten Kommandos aus dem letzten Testskript wenn dabei keine Probleme auftraten. Fehler, die

die Ausführung eines Testskripts zum Abbruch zwingen, werden protokolliert und die Durchführung des Regressionstestes wird mit dem nächsten Skript aus der Auswahlliste fortgesetzt.

3.2.3 Auswertung

Beim Abbruch einer Testsequenz durch Auftreten eines Fehlers wird vom Testsystem deutlich darauf hingewiesen, in der Protokollausgabe für das entsprechende Skript unter \log\ nach der Ursache zu suchen.

```
Testlog
=======

Skript:         Test_AE.1.HTS
Datum:          12.9.2002
Uhrzeit:        13:23

Zeile:          7
HTS-Kommando:   EXISTS,TGT,"DEVELOP.INI"
Ergebnis:       OK...

Zeile:          8
HTS-Kommando:   EXISTS,TGT,"HARDWARE.INI"
Ergebnis:       OK...

...

Zeile:          20
HTS-Kommando:   WAIT,20000
Ergebnis:       OK...

Zeile:          22
HTS-Kommando:   ACTION,MAIN,MENU,CLICK,"Einstellungen","Einstellungen..."
Ergebnis:       OK...

Zeile:          24
HTS-Kommando:   WINDOWEXISTS,"Allgemeine Einstellungen",YES
Ergebnis:       OK...

Zeile:          26
HTS-Kommando:   ACTION,"Allgemeine Einstellungen",EDITBOX,EDIT,"20","Strom"
Ergebnis:       OK...

...

Das nächste HTS-Kommando erstellt zusätzlich die Logdatei:
C:\ATOS_XCTLProjekt\log\Test_AE.1.HTS_12.09.2002_132307_1_DataDiff.log

Zeile:          78
HTS-Kommando:   LAUNCH,BIN,"DATADIFF.EXE","DEVELOP.INI DEVELOP.INI.REF",FOREVER,0,"DataDiff.log"
Ergebnis:       OK...

Zeile:          82
HTS-Kommando:   CLEANUP
Ergebnis:       OK...

Zeile:          84
HTS-Kommando:   DELETE,TGT,"DEVELOP.INI.REF"
Ergebnis:       OK...

Zeile:          85
HTS-Kommando:   COPY,TGT,"DEVELOP.BAK",TGT,"DEVELOP.INI"
Ergebnis:       OK...

Zeile:          86
HTS-Kommando:   DELETE,TGT,"DEVELOP.BAK"
Ergebnis:       OK...

---
Testergebnis:   Das Skript wurde erfolgreich ausgeführt !
```

Abbildung 3.3: Protokolldatei `Test_AE.1.HTS_12.09.2002_132307.log`

Die Protokolldateien, die von *ATOS* erzeugt werden, beschreiben den durchgeführten Testvorgang in semi-verbaler Form. Der Kopf der Dateien enthält den Namen des ausgeführten Testskripts, das Datum, sowie die Uhrzeit der Testdurchführung. Es folgt eine Liste von Einträgen, die den Aktionen des Testskripts entsprechen. Jeder Eintrag umfasst die Zeilennummer, das Kommando und eine verbale Beschreibung. Zusätzlich werden Fehler und sonstige Vorkommnisse protokolliert. Protokolle extern gestarteter Programme wie *DataDiff* werden ebenfalls in das Verzeichnis `\log\` geschrieben. Ein Hinweis dafür wird als Eintrag dem Skript-Protokoll hinzugefügt.

Jede Ausführung eines Testskriptes führt zur Erstellung einer Protokolldatei mit einem eindeutigen Namen. Der Dateiname `Test_AE.1.HTS_12.09.2002_132307.log` gibt zum Beispiel an, dass das Testskript zum Anwendungsfall „Allgemeine Einstellungen“ mit der ID `AE.1` am 12.09.2002 um 13.23 Uhr durchgeführt wurde. Die Protokolldatei `DataDiff.log` wurde vom Dateivergleicher *DataDiff* erstellt und als `Test_AE.1.HTS_12.09.2002_132307_1_DataDiff.log` in das Verzeichnis `\log\` geschrieben. In Abbildung 3.3 wird die Protokolldatei für das Testskript `Test_AE.1.HTS` dargestellt.

3.3 Entwurf von Testfällen

3.3.1 Entwurfsmethodik

Im Laufe des Projektseminars sind zu jedem Anwendungsfall Pflichtenhefte, Verhaltensspezifikationen, Benutzerhandbücher und andere Dokumente entstanden, in denen ihre Funktionalitäten beschrieben werden. Ist die Bedeutung und Benutzung einer Komponente bekannt, können aus den Spezifikation heraus Testfälle abgeleitet werden, mit denen möglichst alle beschriebenen Funktionalitäten mindestens einmal aufgerufen werden. Die Testfälle entsprechen somit repräsentativen Ausführungssequenzen, die ein Benutzer der Steuerungssoftware mit der jeweiligen Komponente durchführen könnte. Auf diese Weise unterliegen die entworfenen Testfälle einer sehr wartungsfreundlichen Ordnung und decken dabei eine Vielzahl der wichtigsten Funktionen des gesamten XCTL-Systems ab.

Es ist leicht vorstellbar, dass alle Kombinationen von Eingabemöglichkeiten für eine Komponente zu unüberschaubaren Mengen von Testfällen führen können. Dazu ein Zitat aus dem Buch „Software automatisch testen“ [1]:

> "...ist mit der Methode, ein Objekt auszuwählen, ein bedeutendes Maß an Zufälligkeit gegeben. Ebenso lassen sich Objekte in unterschiedlicher Reihenfolge auswählen. Innerhalb der Anwendung gibt es generell keine festgelegten Pfade, sondern die Module lassen sich in einer ermüdenden Vielfalt von Reihenfolgen aufrufen und ausführen. Daraus ergibt sich eine Situation, in der Testverfahren nicht ohne weiteres alle möglichen Funktionsszenarien durchspielen können. Die Testingenieure müssen daher ihre Aktivitäten auf den Teil der Anwendung, der die Mehrzahl der Systemanforderung betrifft, und auf die Methoden konzentrieren, wie der Benutzer das System möglicherweise benutzt.“

Beispielsweise ergibt sich bei der „Manuellen Justage“ für nur wenige Antriebe im gleichzeitigen Betrieb in verschiedene Richtungen und unter Beachtung des Verhältnisses der relativen zur absoluten Null bereits ein gewaltiger Kombinationsraum für mögliche Testfälle. Strukturierte Klassifikations-Diagramme helfen hierbei systematisch vorzugehen. Da diese Methode, wie im Kapitel 3.8.1 beschrieben, für komplexe Anwendungsfälle wie „LineScan“ oder „AreaScan“ nicht unproblematisch ist, sollte eine repräsentative Auswahl von Kombinationsmöglichkeiten beim Entwurf von Testfällen angestrebt werden. Im Zuge der Projektarbeit hat sich für dieses Vorgehen der Begriff *Grobtest* heraus gebildet. Für den detaillierten *Feintest* einer bestimmten Teil-Komponente ist ein systematischer Testfall-Entwurf mittels Klassifikationsbaum-Methode durchaus sinnvoll. Beispiele dafür sind im Kapitel 3.8.1 zu finden.

3.3.2 Verwaltung und Organisation

Aufgabe der Testfälle ist die Simulation einer typischen Arbeitsfolge potentieller Benutzer des XCTL-Systems. Außerdem sollen möglichst viele Funktionalitäten einer Komponente mit einer einzigen Testsequenz erfasst werden. Zur Bewältigung dieser Aufgabe wurden zunächst Pflichtenhefte, Verhaltens-Spezifikationen und Benutzerhandbücher aller Anwendungsfälle durchgesehen. Daraus wurden die Aufgaben jedes UseCases herausgearbeitet, um geeignete Aktionsschritte zur Abdeckung der jeweiligen Funktionalitäten in Testsequenzen zusammenzufassen. Ein erstes Ziel war die Erstellung mindestens einer Testsequenz für jeden Anwendungsfall. Wie sich heraus stellen sollte, sind zur Erfassung aller Funktionalitäten für einige Anwendungsfälle sogar nur wenige Testsequenzen ausreichend.

In den meisten Fällen lassen sich die grafischen Benutzerschnittstellen wie Dialogboxen und Menüs den verschiedenen UseCases sehr gut zuordnen. Innerhalb eines Anwendungsfalles sollten die definierten Testfälle im Rahmen zusammengehöriger Dialogboxen und Funktionen operieren. Das unterstützt die Wartung und Übersichtlichkeit der Testfälle und ermöglicht einen isolierten Testlauf für jeden Anwendungsfall. Nicht immer gelingt eine saubere Abgrenzung der Programmfunktionen voneinander. Die Benutzung von Funktionen aus anderen Anwendungsfällen (z.B. Dialogboxen) sollte dennoch nur in Ausnahmefällen stattfinden.

Für umfangreichere Anwendungsfälle, wie „LineScan“ oder „AreaScan“, wurden im Laufe der Projektarbeit weitere Testfälle bzw. Testskripte entworfen, um noch ungeprüfte Funktionalitäten in das Regressionstestpaket aufzunehmen. So wurden die Arbeiten an erweiternden Funktionen, wie beispielsweise den „ContinuousScan“, erst kürzlich abgeschlossen und können nun mit entsprechenden Testfällen für zukünftige Regressionstests berücksichtigt werden. Nach der Hinzunahme weiterer Testfälle, die als Ergebnis der dynamischen Überdeckungsanalyse entstanden (Kapitel 5.1), ist es uns gelungen, eine minimale Überdeckung aller Funktionalitäten im Rahmen der Möglichkeiten der Umgebungssimulation zu erreichen.

Die Testfälle werden zur besseren Wartbarkeit, für alle Projektteilnehmer zugänglich, in das Web-Repository [18] gestellt. Mehrere dieser Testfälle werden zu Testpaketen zusammengefasst und auf jeweils eigenen Webseiten verwaltet. Gültigkeit und Aktualität der Testsequenzen und Referenzdateien für die Komponenten bzw. Anwendungsfälle liegen dabei voll und ganz in der Verantwortung der jeweiligen Bearbeiter. Werden neue Bestandteile in eine Komponente eingearbeitet, bzw. alte Bestandteile überarbeitet, sind die Testfälle auf den aktuellen Stand anzupassen. Alle Testfälle, die aus Spezifikations-Dokumenten hervor gegangen sind, werden im folgenden Abschnitt 3.3.3 beschrieben.

Natürlich bleibt es weiterhin jederzeit möglich, einen Regressionstest bzw. beliebige Testfälle manuell durchzuführen. Dafür müssen alle aktuellen Konfigurations- und Umgebungsdateien aus dem Verzeichnis `\env\` über das CVS-Modul `ATOS_XCTLProjekt` besorgt werden. Für einen Vergleich der Ausgabedateien benötigt man zusätzlich die Referenzdateien aus dem Verzeichnis `\ref\` und das ausführbare Programm `\bin\DataDiff.exe`.

Für den Entwurf der Testfall-Dokumente haben wir die Empfehlungen des IEEE-Standards zur Dokumentation von Software-Tests [13] (ANSI/IEEE Std 829-1983) berücksichtigt. Der Standard ist eine sehr allgemeingültige Empfehlung für jede Art von Tests in beliebigen Software-Entwicklungsphasen. Wie sich heraus gestellt hat, wäre eine Dokumentierung nach dem IEEE-Standard in unserem Fall überdimensioniert. Die komplexe Struktur der Dokumente mit vielen Querverweisen würde zu einer unüberschaubaren und schwer wartbaren Menge an Dokumenten im Web-Repository führen. Der Standard ist nur eine Empfehlung für die Gestaltung der Dokumente und muss an den gegebenen Anwendungsfall angepasst werden.

Zitat aus dem Standard „ANSI/IEEE Std 829-1983“ [13, S.9]:

> „... However, since all of the basic test documents may not be useful in each test phase, the particular documents to be used in a phase are not specified. Each organization using the standard will need to specify the classes of software to which it applies and the specific documents required for a particular test phase.“

Die Empfehlung umfasst die Dokumentation eines Testplans, einer Testspezifikation und einer Testauswertung. Der Testplan ist ein übergeordnetes Dokument, welches die Methodik des Tests beschreibt und die notwendigen Voraussetzungen, Bedingungen und alle zu testenden Softwareobjekte mit ihren untersuchten Gesichtspunkten (Software-Merkmale) auflistet. Es entspricht damit dem Dokument, welches im Web-Repository in der Entwicklertabelle unter Gesamtsystem (Testphase) zu finden ist.

Die Empfehlungen zur Testspezifikation unterscheidet drei verschiedene Dokumenttypen. Die Testdesign-Spezifikation, die Testfall-Spezifikation und die Testverfahren-Spezifikation. Die Abgrenzung der Dokumente in diese drei Typen ist für unseren Fall nicht sinnvoll und würde die Verwaltung und Wartung der Testfälle im Web-Repository nur unnötig schwerfällig machen. Das Dokument zum Testdesign liegt unterhalb des Testplans und verfeinert die Test-Methodik. Zu einem Testplan können mehrere Testdesigns existieren, die wiederum auf einer Menge von Testfall-Spezifikationen verweisen.

Testfall-Spezifikationen definieren Eingaben und erwartete Ausgaben. In den Dokumenten zum Testverfahren werden schließlich die notwendigen Testschritte zur Durchführung eines Testfalles aufgelistet. Da sie neben der eigentlichen Testsequenz auch zusätzliche Voraussetzungen für die Durchführung und Informationen zur Testaufgabe enthalten, entsprechen sie unseren vorgeschlagenen Testfall-Dokumenten im Web-Repository. Zur Vereinfachung wird nur ein einziger Dokumenttyp verwendet, der zusätzlich alle Eingaben und erwarteten Ausgaben enthält.

Die Protokollierung der Testdurchführung findet automatisch durch das Testsystem statt und wird nicht auf Dokumenten im Web-Repository verwaltet. Jede Durchführung eines Regressionstests erzeugt für jeden Testfall eine Reportdatei, die in lesbarer Form zur Auswertung verwendet werden kann. Tritt während der Durchführung eines Skriptes ein Fehler auf, kann die einwandfreie Ausführung der folgenden Aktionsschritte nicht mehr gewährleistet werden. Die Testdurchführung des aktuellen Testfalls wird abgebrochen und beim nächsten Testfall fortgesetzt. Im Protokoll kann leicht die Abbruchstelle ausfindig gemacht werden. Getrennte Dokumente zur Aufzeichnung der Testschritte und dabei aufgetretener Fehler, wie im Standard vorgeschlagen, sind somit in unserem Fall nicht notwendig. Auf Fehler bei einem Regressionstest wird sofort reagiert und Ursachen dafür werden umgehend beseitigt.

Nach erfolgreicher Durchführung eines Regressionstests muss das Datum und die CVS-Marke des „letzten erfolgreichen Durchlaufs“ im Dokumentenkopf aktualisiert werden. Sollen dem Testpaket neue Testfälle hinzugefügt werden, ist die Versionsnummer des Dokumentes hochzuzählen. Der neue Testfall wird mit hochgezählter ID, Namen, Kurzbeschreibung und Sequenztabellen hinter den anderen Sequenzen im Dokument dazugefügt. Neu erstellte Konfigurations-, Umgebungs- und Referenzdateien sowie die Testskripte müssen im CVS-Modul Regressionstest den entsprechenden Ordnern `\env\`, `\ref\`, bzw. `\seq\` hinzugefügt werden.

Die Dokumente zu den jeweiligen Testfällen sind im allgemeinen nach dem folgendem Schema strukturiert:

1. **Dokumentenkopf**
   - Dokumentversion
   - Autor
   - Zustand
   - Letzter erfolgreicher Durchlauf aller Testsequenzen
2. **Aufgabe**
   In diesem Abschnitt wird die Aufgabe des Testpaketes kurz beschrieben. Dabei sind alle Funktionalitäten eines Anwendungsfalles aufzulisten, welche durch die Zusammenstellung geeigneter Testfälle überprüft werden müssen.

3. **Bemerkungen**
   Allgemeine Bemerkungen und zusätzliche Informationen, die für eine Spezifikation von Testfällen wichtig sind, finden hier ihren Platz. Einige Funktionalitäten, wie die Soundausgabe der Detektoren, lassen sich innerhalb der Umgebungssimulation nicht testen. Manchmal muss die Umgebungssimulation sogar „überlistet“ werden, um brauchbare Istwerte für eine Auswertung zu erhalten. Die Eingabedaten müssen dafür bestimmten Richtlinien folgen, um stabile und wiederholbare Ausgaben zu erzeugen. Solche Einschränkungen, die die Möglichkeiten zur Spezifizierung von Testfällen begrenzen, werden in diesem Abschnitt beschrieben.

4. **Testpaket**

   - Testfall 1
     - **ID**
       Eine eindeutige Kennung identifiziert das Testfall-Dokument und korrespondiert mit der zugehörigen Kennung im Dateinamen des Testskriptes.
     - **Skript**
       Dateiname des zugehörigen Skriptes im Verzeichnis \seq\ des ATOS-Projektes im CVS-Repository.
     - **Name**
       Dieser Name sollte für eine möglichst kurze Überschrift des Testfalles benutzt werden.
     - **Kurzbeschreibung**
       In diesem Abschnitt wird der Ablauf der Testsequenz beschrieben. Dabei sollten alle getesteten Funktionalitäten des Anwendungsfalles erwähnt werden.
     - **Vorbereitung**
       Das Vorhandensein wichtiger Konfigurationsdateien und anderer Dateien, die für den Betrieb des XCTL-Systems wichtig sind, muss überprüft werden. Einige Dateien müssen vor der Durchführung der Testsequenz durch präparierte Dateien ausgetauscht werden. Die dazu notwendigen Schritte werden in tabellarischer Form dargestellt. Jeder Schritt (Aktion) wird zum Verständnis in der zugehörigen Spalte (Erklärung) beschrieben. Diese rein verbalen Erklärungen spielen bei der Automatisierung keine Rolle und werden deshalb beim Übergang von den Tabellen zum Testskript nicht beachtet.

  - **Testsequenz**
    Hier findet die Beschreibung der eigentlichen Arbeitsschritte des Benutzers statt. Unabhängige Aktions- und Auswertungsschritte werden zu abhängigen Testschritten zusammengefasst und in Zeilen einer Tabelle aufgelistet. Aktionen bzw. Eingaben stehen Ereignissen bzw. Ausgaben in jeder Zeile gegenüber und finden als Kommandos in den Skripten ihre Entsprechung.
  - **Nachbereitung**
    Nach der Durchführung einer Testsequenz müssen temporär ausgetauschte Umgebungsdateien des XCTL-System wieder in ihren Ausgangszustand zurückgesetzt werden. Nur durch diese Vorgehensweise ist eine Durchführung aufeinanderfolgender und unabhängiger Testsequenzen überhaupt möglich. Auch die Nachbereitung wird in Analogie zur Vorbereitung mit ihren Aktionen und Erklärungen in tabellarischer Form dargestellt.

- Testfall 2
  Beliebig viele weitere Testfälle mit analoger Struktur können folgen bzw. hinzugefügt werden.

- Testfall ...

3.3.3 Entwurf erster Testfälle für Regressionstests

In den folgenden Abschnitten werden 22 Testfälle kurz vorgestellt, die unter Zuhilfenahme von Spezifikations-Dokumenten für die Anwendungsfälle entworfen wurden. Auf eine vollständige Wiedergabe der Testfall-Dokumente wurde dabei verzichtet. Vorgestellt werden die grundsätzlichen Aufgaben der Testpakete und ihrer Testfälle. Die Umgebungssimulation und das Verhalten des XCTL-Systems selbst beschränken die Möglichkeiten zur Spezifikation von Testfällen. Die Konsequenzen für den Entwurf von Test-Szenarien werden in den folgenden Abschnitten unter *Bemerkungen* dokumentiert. In den Dokumenten im Web-Repository [18] sind die vollständigen Sequenzen-Tabellen für jeden beschriebenen Testfall einzusehen. Beispielhaft ist die Tabelle mit den durchzuführenden Testschritten für den Testfalles MJ.1 aus dem Anwendungsfall „Manuelle Justage“ im Anhang B.1.

Anwendungsfall „Motorsteuerung“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Motorsteuerung über die dafür bereitgestellten Dialogboxen. Die Motorsteuerung enthält Kernfunktionen zur Initialisierung, Ansteuerung und Verwaltung von Motoren im XCTL-Programm. Die Testfälle müssen dabei Funktionen zum Referenzpunktlauf mit Anfahren der Grundstellung (bzw. Ausgangsstellung), Funktionen zur Kalibrierung sowie Funktionen zur Einstellung und Optimierung der Motorparameter überprüfen. Auch die Möglichkeit zur direkten Positionierung der Antriebe über interne Schritte (Encoder-Positionen) darf dabei als Funktion nicht übersehen werden.

Bemerkungen

Im Rahmen der Umgebungssimulation kann mit dieser Testfallsammlung nur die Kommunikation mit der Motorensimulation (`msim.dll`) überprüft werden. Reale Motoren-Controller sind nicht ansprechbar. Die zugehörigen Funktionalitäten dieses Anwendungsfalles lassen sich dennoch sehr gut mit der Simulation testen.

Die Testfälle zur Überprüfung dieses Anwendungsfalles erfordern häufig das Umschalten zwischen unterschiedlichen Dialogboxen. Eine saubere Zuordnung der Dialogboxen und Fenster zu den untersuchten Funktionalitäten, wie in den Testfällen anderer Anwendungsfälle, läßt sich daher schwer realisieren. Viele Ergebnisse von Eingaben in Dialogboxen werden erst in anderen Dialogboxen sichtbar. Bis auf das „LineScan“-Fenster, das bei der „Optimierung“ seine Anwendung findet, werden dennoch nur Dialogboxen verwendet, die dem Anwendungsfall Motorsteuerung zuzuordnen sind, womit sich auch dieser Anwendungsfall relativ isoliert von den anderen untersuchen läßt.

Einige Parameter, die in der `HARDWARE.INI` oder in den Dialogboxen eingestellt werden, beeinflussen nicht die Arbeit der Antriebe innerhalb der Umgebungssimulation. Da ihre Werte keine auswertbaren Ergebnisse hervorrufen, spielen sie für die Spezifikation von Testfällen keine Rolle und können beliebig gesetzt werden, sofern das die jeweilige Eingabemaske zuläßt. Für die Dialogboxen betrifft das die Parameter P-Filter (DynamicGain), I-Filter (IntegralGain), L-Filter (IntegralLimit), D-Filter (Gain) bei C-832 Antrieben sowie Gain, Torque und DynamicGain bei C-812 Antrieben.

Der Vorgang des „Optimierens“ findet normalerweise nur über Kurvenausgaben auf dem Bildschirm statt. Um den automatisierten Test dieser Funktion zu unterstützen, wird zusätzlich die Beschleunigungskurve als `*.bk` Datei abgespeichert und mit einer Referenzdatei verglichen.

Testfälle

Testfall 1
ID: MS.1
Name: Grundstellung Anfahren für zwei Antriebe
Kurzbeschreibung:
Die Dialogboxen „Grundstellung anfahren“ und „Verfahren nach Encoder-Position“ werden benutzt. Der Antrieb DF wird auf seinen absoluten Nullpunkt zurück gesetzt, auf seine Indexposition und schließlich auf seine Grundstellung gefahren. Über die „Direkte Steuerung“ wird der Antrieb DF wieder auf 0 gefahren. Der Antrieb Tilt wird neu kalibriert und seine Position über die „Direkte Steuerung“ ausgelesen.

Testfall 2
ID: MS.2
Name: Einstellung und Optimierung von Antriebsparametern
Kurzbeschreibung:
Die Dialogboxen „Grundstellung anfahren“ und „Verfahren nach Encoder-Position“ werden benutzt. Der Winkelbereich des Antriebs „Azimutal Rot“ wird verkleinert und durch die Funktion „Bereichsmaximierung“ wieder auf seinen ursprünglichen Wert gesetzt. Die Antriebe „Azimutal Rot“ und „DF“ werden danach mit den entsprechenden Dialogboxen optimiert, wobei jeweils eine Beschleunigungskurve als `*.bk` Datei abgespeichert wird.

Anwendungsfall „Detektornutzung“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Benutzung von 0-dimensionalen Detektoren. Dabei sollen die Funktionen zur Auswahl und Einstellung der Detektoren, zur Darstellung der Messergebnisse sowie zur Protokollierung der Intensitäten im Zählerfenster und in einer Datei durch geeignete Testfälle überprüft werden.

Bemerkungen

Die Dialogbox zur Zähler-Konfiguration ist für die Verwaltung jeglicher Detektoren, also auch 1-dimensionaler (bzw. 2-dimensionaler) Zähler verantwortlich. Ein gesondertes Vorgehen für die Überprüfung der Funktionalitäten für einen 1(2)-dimensionalen Detektor ist somit nicht nötig.

Es ist zu beachten, dass mit diesen Testfällen keineswegs die tatsächliche Funktionsweise eines angeschlossenen Detektors, sondern nur die Möglichkeiten seiner Verwendung innerhalb des XCTL-Systems getestet werden. Da nur der 0-dimensionale Detektor „Radicon“ erweiterte Einstellungsmöglichkeiten für Energieschwellwerte und Spannungen bietet, die innerhalb der Umgebungssimulation jedoch nicht getestet werden können, genügt der Testdetektor von K. Schützler zur Validierung der grundsätzlichen Programmfunktionalitäten zur Benutzung eines 0-dimensionalen Detektors. Die Optionen Messfehler und Soundausgabe haben keinen Bedeutung bei der Verwendung von Testdetektoren.

Aufgrund eines Programmfehlers ist es leider (noch) nicht möglich, mit verschiedenen 0-dimensionalen Testdetektoren gleichzeitig zu arbeiten. Es wird immer nur die aktuelle Intensität des ersten Testdetektors aus der Konfigurationsdatei `HARDWARE.INI` im Zählerfenster dargestellt. Weiterhin ist die Benutzung von Detektoren durch das XCTL-System auf die maximale Anzahl von drei beschränkt.

Testfälle

Testfall 1
ID: D0.1
Name: Testdetektor mit Digitaldarstellung und Protokollabbruch
Kurzbeschreibung:
Die Dialogbox „Zähler-Konfiguration“ wird aufgerufen. Das Zählerfenster wird mit Digitalanzeige aufgerufen und geschlossen. Das Protokoll wird gestartet und abgebrochen. Detektoreinstellungen werden getroffen und wieder verworfen.

Testfall 2
ID: D0.2
Name: Testdetektor mit Balkendarstellung und Protokollunterbrechung
Kurzbeschreibung:
Das Zählerfenster wird mit Balkenanzeige aufgerufen und geschlossen. Das Protokoll wird gestartet und kurzzeitig unterbrochen.

Anwendungsfall „Ablaufsteuerung“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Arbeit mit Makrodateien. Das Laden, Löschen, Ausführen, Anhalten und Fortsetzen von Makros, sowie die Protokollierung auf dem Bildschirm und in einer Datei sollen dabei als essentielle Funktionen dieses Anwendungsfalles überprüft werden.

Bemerkungen

Die derzeitige XCTL-Version ermöglicht nur die Einbindung und Ausführung der vorgefertigten Makros „SetupTopography“, „SearchReflection“, „InquireHwb“, „AzimutalJustify“ und „Test“ aus der Datei `STANDARD.MAK`, sowie der zwei Makros „ScanJob“ und „AreaScanJob“ aus der Datei `SCAN.MAK`. Selbst erstellte Makros können nicht geladen werden, so dass man darauf angewiesen ist, die vorhandenen Makrodateien für einen Testlauf zu verwenden. Es genügt hierfür das Makro „Test“, um repräsentative Befehlsfolgen zu ergänzen, so dass ein einziger Aufruf dieses Makros die Funktionsweise der Makroausführung ausreichend testet und zu vergleichbaren Ergebnissen führt. Verständlicher Weise können nicht alle Kombinationen von Makrobefehlen mit ihren unterschiedlichen Auswirkungen auf das XCTL-System getestet werden. Daher beschränkt man sich auf eine repräsentative Befehlsfolge, die zum Positionieren der Motoren und zur Messung der Halbwertsbreite notwendig sind.

Der zuständige Algorithmus zur Bestimmung der Halbwertsbreite sieht normaler Weise eine Schrittweite von 0.4 Sekunden für den Antrieb „DF“ vor. Die Auflösung der virtuellen Testprobe `TESTDEV.DAT` für den Antrieb „DF“ beträgt jedoch 1.0 Sekunden, wodurch es bei wiederholten Messungen zu abweichenden Ausgaben in der Protokolldatei `MACRO.LOG` kommt. Der zufällige Anteil des 0-dimensionalen Detektors beeinflußt dabei das Suchverhalten des Algorithmus' so, dass einmal mehr oder weniger Schritte zum Auffinden eines Peaks benötigt werden. Eine unterschiedliche Anzahl an Protokolleinträgen ist die Folge. Für einen automatisierten Vergleich mit einer Solldatei muss jedoch die Anzahl aller Einträge von Intensitäten und Motorpositionen übereinstimmen. In der präparierten Datei `TEST_STANDARD.MAK` wurde deshalb die Schrittweite auf 1.0 festgelegt.

Die Ausführungsfähigkeit von Makros wird eigentlich bei vielen anderen Anwendungsfällen eingesetzt, weshalb die redundante Überprüfung dieser Funktionalität kritisiert werden könnte. Beispielsweise wird beim Anwendungsfall „Halbwertsbreite messen“ das Makro „InquireHwb“ aus der Datei `STANDARD.MAK` ausgeführt. Da die „Ablaufsteuerung“ innerhalb des Projektes jedoch als eigenständiger Anwendungsfall heraus gearbeitet wurde, soll auch hierfür mindestens eine Testsequenz entworfen werden, um das Ausführen von Makros unabhängig von anderen Anwendungsfällen zu überprüfen.

Testfälle

Testfall 1
ID: AS.1
Name: Anspringen bestimmter Motorpositionen und Messen der HWB
Kurzbeschreibung:
Die Dialogbox zur Ausführung von Makros wird aufgerufen, das Testmakro ausgewählt und ausgeführt. Die Motoren werden auf bestimmte Positionen bewegt. Von dort aus wird die Halbwertsbreite bestimmt. Alle gemessenen Intensitäten des 0-dimensionalen Testdetektors werden protokolliert.

Anwendungsfall „Topographie“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zum „Einstellen der Parameter“ sowie zum „Start und Kontrolle“ als Teile des Topographie-Gesamtvorganges. Die beiden Messmethoden „Topographie mit Einfachbelichtung“ und „Topographie mit Mehrfachbelichtung“ mit ihren Funktionen zum Anfahren des Arbeitspunktes, zur abnormalen Ausnahmebehandlung, zum Starten und Stoppen der Regelung (bzw. Topographie) und zur Ausgabe des Vorganges in der Dialogbox sollen dabei von entsprechenden Testfällen untersucht werden. Die Parameter beider Messmethoden sind dafür in einer Dialogbox festzulegen.

Bemerkungen

Ein getrennter Test für die Teilbereiche „Einstellen der Parameter für die Topographie“ und „Start und Kontrolle der Topographie“ wäre nicht sehr sinnvoll. Die Durchführung der Topographie stützt sich zu großen Teilen auf Parameter, die in der Dialogbox „Einstellungen Topographie“ vorgenommen wurden. Von der derzeitigen Programmversion kann diese Konfiguration jedoch nicht dauerhaft in die Datei `DEVELOP.INI` übernommen werden. Diese Tatsachen führen zu der Einsicht, beide Teilbereiche gemeinsam zu testen.

Die eigentlichen Ergebnisse der Funktionalitäten zur Topographie sind innerhalb der Umgebungssimulation nicht sichtbar, da Fotoplatten und Filme nicht ausgewertet werden können und die Einbindung eines 2-dimensionalen CCD-Detektors in das XCTL-System vorläufig noch nicht vollzogen wurde. Die Ergebnisse des 0-dimensionalen Testdetektors, die Statusausgaben der Dialogbox und die verbrauchten Meßzeiten stellen somit die einzigen auswertbaren Resultate dieser Komponente dar. Zur Überprüfung der beiden Teilbereiche Einstellungen und Kontrolle der Topographie sind die zusammengestellten Testfälle jedoch vollkommen ausreichend.

Zu einer automatischen Nachregelung des Antriebs „DF“ aufgrund thermisch verursachter Intensitätsschwankungen kann es innerhalb der Umgebungssimulation nie kommen. Um dieses Verhalten zu simulieren, müssten sich die Intensitätswerte des 0-dimensionalen Detektors spontan zur Laufzeit verändern können. Somit wird auch eine abnormale Ausnahmebehandlung für den Fall mit Einfachbelichtung innerhalb dieser Testfälle nicht auftreten können, da die fehlende Nachregelung niemals zum Herauslaufen aus festgelegten Grenzen (z.B. Beschränkung auf ± 50.0 Sekunden) führen kann.

Akustische Signale zur Kennzeichnung des Abschlusses eines Messvorgangs werden nur über die reale Controller-Hardware ausgegeben und können bei der Spezifizierung von Testfällen innerhalb der Umgebungssimulation nicht berücksichtigt werden.

Aus der Dialogbox zur Kontrolle der Topographie gelangt man direkt zur Einstellungen-Dialogbox, welche jedoch in der aktuellen XCTL-Version nicht für alle Parameter Änderungen zuläßt. Zum Test dieser Funktionalität empfiehlt sich deshalb immer der Weg über das Hauptmenü.

Testfälle

Testfall 1
ID: TP.1
Name: Topographie mit Einfachbelichtung
Kurzbeschreibung:
Die Dialogbox zur Festlegung der Topographieparameter wird aufgerufen und mit Werten belegt. Danach wird die Startposition (Arbeitspunkt auf der linken Flanke der Rocking-Kurve) mittels des Makros „SetupTopography“ angefahren und der Messvorgang in festgehaltener Motorposition vollzogen.

Testfall 2
ID: TP.2
Name: Topographie mit Mehrfachbelichtung
Kurzbeschreibung:
Die Dialogbox zur Festlegung der Topographieparameter wird aufgerufen und mit Werten belegt. Danach wird die Startposition durch ein programminternes Makro direkt angefahren (nicht über den Peak wie bei Einfachbelichtung) und der Messvorgang in drei Segmenten durch schrittweises Anfahren bestimmter DF-Motorposition vollzogen. Jedes Segment wird dabei für 1 Minute ruhig gehalten und ausgeleuchtet. Dies entspricht der Methode der Mehrfachbelichtung, wie sie bei abnormalen Probenverformungen angewendet wird.

Anwendungsfall „Diffraktometrie/Reflektometrie“

Der Anwendungsfall Diffraktometrie/Reflektometrie wird in die beiden Teilbereiche „LineScan“ und „AreaScan“ unterschieden. Für beide Messmethoden stehen jeweils verschiedene Darstellungsfenster und Einstellungsdialoge zur Verfügung, weshalb für jede Methode ein eigenes Testpaket zusammengestellt wird.

LineScan

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Durchführung eines LineScans als Teil des Diffraktometrie/Reflektometrie-Gesamtvorganges. Die Funktionalitäten dieses Anwendungsfalles sind sehr vielfältig. Der Bereich „Einstellungen und Ablauf des LineScans“ umfasst dabei im wesentlichen Funktionen zur Erfassung, Sicherung und Darstellung von Messwerten eines 0-dimensionalen Detektors.

Jüngste Überarbeitung dieser XCTL-Komponente führten zu Erweiterungen um die Funtionen „ContinuousScan“ und „Dynamische Schrittweite“. ContinuousScan und StepScan als spezielle LineScans unterscheiden sich durch kontinuierliche bzw. schrittweise Bewegung der Probe und sind daher mit jeweiligen Testfällen zu überprüfen

Bemerkungen

Da der Theta-Winkel bei der Messung der Intensitäten keine Rolle spielt, erreicht man eine unterschiedliche Kurve innerhalb des Omega-Bereichs, ohne Veränderung der anderen Antriebe, nur durch Modifikation der Schrittweite. Eine feinere Kurve mit kleinen Schwankungen, die sich sichtbar von der groberen Kurve abhebt, ist die Folge und für einen Vergleichs-Scan geeignet.

Die Hardware-Konfiguration für die Diffraktometrie/Reflektometrie-Arbeitsplätze enthalten normalerweise keine Einträge für einen Kollimator. Da der 0-dimensionale Testdetektor zur Ermittlung der Intensität mit Hilfe der Testprobe `TESTDEV.DAT` jedoch die aktuelle Stellung dieses Antriebs berücksichtigt, ist die Einbindung des Kollimators in die `HARDWARE.INI` vorteilhaft, um sinnvolle und auswertbare Kurven zu erhalten.

Zur Kontrolle der Messung mittels eines weiteren Detektors (Monitor) darf auch innerhalb der Umgebungssimulation ein beliebiger Testdetektor, einschließlich der 1-dimensionale, ausgewählt werden. Die Monitordaten werden zur Normierung für die aktuell gemessene Intensität verwendet, um daraus die nächste Schrittweite bei einer dynamischen Schrittweitensteuerung zu bestimmen.

Die Funktion des „ContinuousScan“ läßt sich derzeit nur mit dem 0-dimensionalen Detektor „Radicon“ und dem 0-dimensionalen Testdetektor „Simulant“ von Hr. Damerow durchführen. Die Testdetektoren „Counter“ und „PSD“ wurden für diese Funktion noch nicht erweitert und können deshalb nicht für Testfälle eingesetzt werden. Für einen Test innerhalb der Umgebungssimulation ist somit die Wahl auf den 0-dimensionalen Testdetektor Simulant von Hr. Damerow beschränkt. Dabei führt jede Messungsunterbrechung zum Neustart des implementierten Simulations-Algortihmus', weshalb eine Unterbrechung in den Testfällen nicht vorgenommen werden sollte.

Testfälle

Testfall 1
ID: LS.1
Name: Vergleich zweier verschiedener StepScans
Kurzbeschreibung:
Das LineScan-Fenster wird aufgerufen. Die Testprobe (`TESTDEV.DAT`) wird zunächst über Omega-2Theta von einem Start- zu einem Endwinkel mit bestimmter Schrittweite gescannt und dabei kurzzeitig unterbrochen. In einem zweiten Scan wird nur über die Omega-Achse mit einer anderen Schrittweite und unter Beobachtung eines weiteren Detektors als Monitor gescannt. In jedem dieser Schritte wird die von der Probe reflektierte Röntgenstrahlung mit Hilfe des 0-dimensionalen Testdetektors gemessen. Diese über den jeweiligen Winkeln abgetragenen Intensitäten werden in Form von Kurven dargestellt und als `*.crv` Dateien abgespeichert. Die zweite fixierte Kurve wird außerdem als `*.bk` Datei abgespeichert.

Testfall 2
ID: LS.2
Name: ContinuousScan
Kurzbeschreibung:
Das LineScan-Fenster wird aufgerufen. Ein ContinuousScan wird mit dem 0-dimensionalen Testdetektor von Herrn Damerow durchgeführt. Die über den jeweiligen Winkeln gemessenen Intensitäten werden in Form einer Kurve dargestellt und als `*.crv` Dateien abgespeichert.

Testfall 3
ID: LS.3
Name: StepScan mit dynamischer Schrittweite
Kurzbeschreibung:
Das LineScan-Fenster wird aufgerufen. Die Testprobe (`TESTDEV.DAT`) wird über die Omega-Achse von einem Start- zu einem Endwinkel mit dynamischer Schrittweite gescannt und dabei kurzzeitig unterbrochen. In jedem dieser Schritte wird die von der Probe reflektierte Röntgenstrahlung mit Hilfe des 0-dimensionalen Testdetektors gemessen und in Abhängigkeit davon die aktuelle Schrittweite angepasst. Diese über den jeweiligen Winkeln abgetragenen Intensitäten werden in Form einer Kurve dargestellt und als `*.crv` Datei abgespeichert.

AreaScan

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Durchführung eines AreaScans als Teil des Diffraktometrie/Reflektometrie-Gesamtvorganges. Die Funktionalitäten dieses Anwendungsfalles sind sehr vielfältig. Der Bereich „Einstellungen und Ablauf des AreaScans“ umfasst dabei im wesentlichen Funktionen zur Erfassung, Sicherung, Darstellung und Auswahl von Messwerten eines 0- oder 1-dimensionalen Detektors.

Da der PSD-Detektor im starken Zusammenhang zum AreaScan steht und die Funktionalitäten „Kalibrierung“, „Energie-Spektrum anzeigen“ und „Einstellungen“ nur über das Kontextmenü des „AreaScan“-Fensters erreichbar sind, werden die erwähnten Funktionen innerhalb dieses Anwendungsfalles überprüft.

Jüngste Überarbeitung dieser XCTL-Komponente führten zu Erweiterungen um folgende Funtionen: „Omega, Thata- und Psd-Kanal-Offset“, „Datenbasisvergrößerung und Zerlegung von PSD-Dateien“, „Zusatzinformationen zu Spektren der Datenbasis“ sowie „Akkumulierte Spektrenanzeige“.

Alle aufgeführten Funktionalitäten sind durch entsprechende Testfälle zu überprüfen.

Bemerkungen

Da der Theta-Winkel bei der Messung der Intensitäten keine Rolle spielt, ergibt die Kurve des SLD-Scans mit dem 0-dimensionalen Detektor nur eine Gerade mit leichten Schwankungen bei jeder Omega-Stellung. Trotzdem die visuelle Ausgabe einer realen Messung damit nicht sehr nahe kommt, kann die reine Funktionalität eines AreaScans mit einem 0-dimensionalen Detektor dennoch innerhalb der Umgebungssimulation überprüft werden.

Die Erweiterung „Zusatzinformationen“, welche die ehemaligen Reportdateien um einige Einträge ergänzen, müssen nicht gesondert überprüft werden, wenn im Zuge der Testfälle zum AreaScan unterschiedliche Optionen zur Ausgabe (Monitor, Absorber etc.) eingestellt werden.

Die Darstellung des Energie- und Impulsspektrums ist für den 1-dimensionalen Testdetektor identisch. Der in der Realität auftretende Unterschied ist innerhalb der Umgebungssimulation nicht überprüfbar. Das Impulsspektrum unterliegt, abhängig vom Substrat, zum Teil starken Schwankungen, während das Energiespektrum, abhängig von der Strahlungsquelle, eine Kurve liefert, die ihr Maximum immer bei den gleichen Kanälen hat.

Ein Test der PSD-Kalibrierung ist innerhalb der Umgebungssimulation möglich, entspricht jedoch nicht den Werten realer 1-dimensionaler Detektoren. Die Funktion der Kalibrierung beinhaltet das Verschieben des Peaks durch Bewegen der Thetaachse vom ersten bis zum letzten Kanal (bzw. umgekehrt). Die Peakposition ändert sich nur minimal für große Theta-Distanzen, so dass entsprechend große Werte für die Winkel ermittelt werden, die einem Kanal des PSD entsprechen. Für die Überprüfung der Funktionalität dieser Komponente ist das jedoch völlig ausreichend.

Nach Durchlauf eines AreaScans, führt das Fixieren und Nachladen einer abgespeicherten Kurve nicht zum gewünschten Resultat. In der aktuellen XCTL-Version wird die fixierte unter der nachgeladenen Kurve verzerrt dargestellt. Für einen Test dieser Funktionalität ist daher eine Reinigung des Areascan-Fensters mittels „Datei → Neu“ aus dem Hauptmenü vor dem Nachladen beider Kurven notwendig.

Testfälle

Testfall 1
ID: AS.1
Name: Vergleich zweier verschiedener AreaScans
Kurzbeschreibung:
Das AreaScan-Fenster wird geöffnet. Zwei PSD-Messungen mit verschiedenen Parameter-Einstellungen werden vorgenommen und anschließend miteinander verglichen. Der erste Scan wird im Omega2Theta-Modus mit PSD-Offset durchgeführt und anschließend mittels der Funktion „Scanauswahl“ untersucht. Der zweite Scan wird mit festgehaltener Theta-Achse im Standard-Modus mit akkumulierter Spektrenanzeige durchgeführt und zwischenzeitlich unterbrochen. Es entstehen zwei Messwertdateien (`*.psd`), zwei Reportdateien (texttt*.rep) sowie eine Vergleichs-Scandatei (`*.bk`).

Testfall 2
ID: AS.2
Name: Areascan mit 0-dimensionalen Detektor (SLD-Scan)
Kurzbeschreibung:
Das AreaScan-Fenster wird geöffnet. Eine Messung mit Parameter-Einstellungen für den 0-dimensionalen Testdetektor wird vorgenommen. Für Omega und Theta werden dabei Offsets festgelegt. Die erhaltenen Messwerte werden in `*.crv` Dateien zerlegt und zu einer kürzeren Datei zusammengefügt, so dass wiederum `*.psd`, bzw. `*.rep` Dateien entstehen.

Testfall 3
ID: AS.3
Name: Arbeit mit dem 1-dimensionalen Detektor (PSD)
Kurzbeschreibung:
Das AreaScan-Fenster wird geöffnet. Das Impuls-Spektrum und das Energie-Spektrum werden angezeigt. Der PSD wird kalibriert. Die Einstellungen für den 1-dimensionalen Testdetektor werden verändert.

Anwendungsfall „Darstellung von Messdaten“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur „Bildnerischen Darstellung der Messergebnisse bei der Diffraktometrie/Reflektometrie“. Während bei einem LineScan nur die Darstellungsart „Curve“ zur Verfügung steht, ist bei einem AreaScan die Auswahl zwischen „RL-Bitmap“, „Raw Matrix“ und „Curve“ möglich. Innerhalb der Dialogbox sind viele weitere Einstellungen zur Festlegung der grafischen Ausgabe möglich. Dort kann die Auflösung, Beschriftung und Einteilung der Achsen, das Intensitäts-Maximum (bzw. Minimum) sowie die Intensitäts-Skalierung eingestellt werden. Da die Funktion der Datenerhebung nur innerhalb der Darstellung als „RL-Bitmap“ bzw. „Raw Matrix“ durchführbar ist, wird sie innerhalb dieses Anwendungsfalles behandelt.

Bemerkungen

Der Anwendungsfall „Bildnerischen Darstellung der Messergebnisse“ greift auf fundamentale Teilfunktionen des Anwendungsfalles „Diffraktometrie/Reflektometrie“ zu. Die Fenster für einen AreaScan bzw. LineScan mit ihren Konextmenüs sowie Funktionen zum Einlesen von Daten werden für einen Test benötigt. Aus diesem Grund ist es unmöglich, die Funktionen zu diesem Anwendungsfall isoliert von anderen zu überprüfen.

Um sinnvolle Kurvengrafiken zu erhalten, bietet sich die Verwendung einer Messwertedatei aus realen Bedingungen an. Zu diesem Zwecke steht die Datei `m4680.psd` zur Verfügung, die am 04.01.1980 an einem Diffraktometrie-Arbeitsplatz aufgezeichnet wurde. Eine passende Reportdatei `m4680.rep` existiert leider nicht, weshalb das XCTL-Programm beim Einlesen der Messdaten eine Warnung ausschreibt. Zu beachten ist hierbei, dass das Einlesen von Messwertdateien die nutzerspezifischen Angaben (Name, Probe etc.) überschreibt.

Testfälle für diesen Anwendungsfall lassen sich nur schlecht automatisieren. Alle ausgegebenen Grafiken müssen vom Tester mit den Referenzbildern selbständig verglichen werden. Auch die Arbeit mit der Maus kann mit dem Testsystem nicht automatisiert vollzogen werden, weshalb die Funktionalität „Datenerhebung“ ebenfalls durch manuellen Eingriff des Testers überprüft werden muss.

Da sich die grafische Darstellung der Meßergebnisse im LineScan-Modus (ausschließlich Darstellungsart „Curve“ verfügbar) nicht von der grafischen Darstellung beim AreaScan-Modus in der Darstellungsart „Curve“ unterscheidet, ist der Entwurf eines gesonderten Testfalles nicht notwendig. Das Nachladen und Visualisieren von `*.crv` bzw. `*.bk` Dateien wird indirekt durch die Testfälle zum Anwendungsfall Diffraktometrie/Reflektometrie (LineScan) überprüft.

Testfälle

Testfall 1
ID: DM.1
Name: Darstellung der Messergebnisse bei der Reflektometrie
Kurzbeschreibung:
Das AreaScan-Fenster wird geöffnet. Eine `*.psd` Datei wird eingelesen und durch Verändern der Darstellungsparameter auf unterschiedliche Weise visualisiert. Hierbei müssen stets die dargestellten Ergebnisse mit den angegebenen Referenzbildern verglichen werden!

Testfall 2
ID: DM.2
Name: Datenerhebung bei der Reflektometrie
Kurzbeschreibung:
Das AreaScan-Fenster wird geöffnet. Eine `*.psd` Datei wird eingelesen und als RAW-Matrix visualisiert. Eine Datenerhebung wird durchgeführt und als `*.dtn` Datei gespeichert. (Benutzung der Maus auf dem Bitmap)

Anwendungsfall „Automatische Justage“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten der automatischen Probenjustage für die Topographie. Im Statusfenster werden die Schritte der Justage mit den angesteuerten Motorpositionen, den jeweiligen Röntgenintensitäten und der Justagedauer ausgegeben. Auf Wunsch kann das Protokoll auch in der Datei `Justage.log` abgelegt werden. Der Algorithmus zur Bestimmung der Antriebspositionen läßt sich durch Angaben über die Anzahl der Durchläufe und Intensitätsmessungen sowie über den Suchbereich in der zuständigen Dialogbox beeinflussen.

Ist die automatische Justierung abgeschlossen, liegt die erreichte Intensität niemals unterhalb der Ausgangsintensität. Alle aufgeführten Funktionen sind mit geeigneten Testfällen zu überprüfen.

Bemerkungen

Der Algorithmus zum Auffinden des Intensitätsmaximums arbeitet innerhalb der Umgebungssimulation leider nicht deterministisch. Jeder Durchlauf einer automatischen Justierung führt aufgrund der groben Messdaten-Auflösung in der Datei `TESTDEV.DAT`, im Zusammenspiel mit dem Zufallsanteil des 0-dimensionalen Testdetektors, zu unterschiedlichen Intensitäten und Antriebspositionen. Sicher gestellt ist nur, dass niemals ein niedrigerer Wert als die Ausgangsintensität erreicht wird.

Beispiel: Die Intensitätswerte ändern sich für den Kollimator nur alle 70 μm. Der Algorithmus springt jedoch zwischen diese Abstände und erhält somit vom Testdetektor dieselben Intensitätswerte, der um einen zufälligen Anteil schwankt. Je nachdem, ob der zufällige Anteil den vorherigen Wert über- oder unterschreitet, bewegt sich der Kollimator in eine unterschiedliche Richtung weiter.

Die Anzahl der Einträge „Optimieren“ und „Nachregeln“ der jeweiligen Antriebe bleibt trotz unterschiedlicher Suchabläufe in den Protokolldateien konstant. Eine grobe Überprüfung der Funktionalität dieses Anwendungsfalles ist somit auch im Rahmen der Umgebungssimulation möglich.

Testfälle

Testfall 1
ID: AJ.1
Name: Automatische Justage mit Ausgabe in eine Logdatei
Kurzbeschreibung:
Die Dialogbox „Automatische Justage“ wird aufgerufen und die Abbruchkriterien werden festgelegt. Nach Start der automatischen Probenjustierung stellt der Algorithmus die Antriebe auf eine Position mit maximaler Intensität im Rahmen des vorgegebenen Suchbereichs ein.

Anwendungsfall „Manuelle Justage“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Ansteuerung von Antrieben („DF“, „DC“, „AR“, „TL“, „CC“ etc.). über die Dialogbox „Manuelle Justage“. Die Motoren sollen hierbei mit den drei Betriebsarten Direktbetrieb, Schrittbetrieb und Fahrbetrieb angesteuert werden können. Alle Antriebe, bis auf

„DF“ und „DC“, können dabei unabhängig voneinander bedient werden und gleichzeitig in Bewegung sein. Die Ein und -Ausgaben der Winkel beziehen sich auf die relative Null-Position, welche für jeden Antrieb beliebig festgelegt werden darf. Für den funktionellen Test dieser Dialogbox genügen Testfälle zum Umgang mit den drei wichtigsten Antrieben „DF“ („Beugung Fein“), „TL“ („Tilt“) und „CC“ („Collimator“).

Bemerkungen

Bei der Spezifizierung von Testfällen ist zu beachten, dass häufige Richtungsänderungen der Antriebe zu Rundungsfehlern bei der Positionsbestimmung führen können. Diese Tatsache ist gegebenenfalls durch jeweilige Toleranzen zu berücksichtigen.

Während der Bewegung eines Antriebs ist die Auswahl eines anderen über die Kurzwahl nicht möglich. Die Auswahl kann aber über die Auswahlliste unter „Aktueller Antrieb“ vorgenommen werden.

Testfälle

Testfall 1
ID: MJ.1
Name: DF im Direktbetrieb und Setzen der „Relativen Null“
Kurzbeschreibung:
Die Dialogbox „Manuelle Justage“ wird aufgerufen. Der Antrieb „DF“ wird ausgewählt und im Direktbetrieb auf bestimmte Werte bewegt. Das Setzen und Aufheben der relativen Null wird hierbei getestet.

Testfall 2
ID: MJ.2
Name: Tilt im Schrittbetrieb
Kurzbeschreibung:
Die Dialogbox „Manuelle Justage“ wird aufgerufen. Der Antrieb Tilt wird ausgewählt und im Schrittbetrieb auf bestimmte Werte bewegt.

Testfall 3
ID: MJ.3
Name: Kollimator im Fahrbetrieb
Kurzbeschreibung:
Die Dialogbox „Manuelle Justage“ wird aufgerufen. Der Antrieb Kollimator wird ausgewählt und im Fahrbetrieb auf bestimmte Werte bewegt.

Testfall 4
ID: MJ.4
Name: Kollimator und DF gleichzeitig im Direktbetrieb
Kurzbeschreibung:
Die Dialogbox „Manuelle Justage“ wird aufgerufen. Der Kollimator-Antrieb wird ausgewählt und im Direktbetrieb auf einen Wert bewegt. Gleichzeitig wird der Antrieb „DF“ im Direktbetrieb auf einen Wert gefahren.

Anwendungsfall „Allgemeine Einstellungen“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Festlegung der Betriebsparameter und Probenangaben. Benutzerangaben können über eine Dialogbox abgelesen, eingestellt und verworfen werden. Nach Programmende werden diese Benutzerangaben im Abschnitt `[Steuerprogramm]` der Datei `DEVELOP.INI` abgelegt. Die alten Benutzerangaben werden dabei überschrieben. Nach dem Programmstart sollten alle aktuellen Benutzerangaben in der Dialogbox „Allgemeine Einstellungen“ vorzufinden sein. Dieses Verhalten ist mit geeigneten Testfällen zu überprüfen.

Bemerkungen

Die Benutzerangaben haben keinen Einfluss auf das Steuerungs- und Messverhalten des XCTL-Systems. Es sind rein informative Beschreibungen über den Messplatz, den Benutzer und die untersuchte Probe. Anwendung finden sie beim Erstellen und Einlesen von unterschiedlichen Messwertedateien und werden dort als zusätzliche Informationen in den Headern verwaltet. Das Einlesen einer solchen Datei aktualisiert entsprechend die aktuellen Benutzerangaben. Das Schreiben bzw. Einlesen von Headerinformationen wird in den Testfällen anderer Anwendungsfälle indirekt benutzt und untersucht.

Testfälle

Testfall 1
ID: AE.1
Name: Modifizierung von Benutzerangaben
Kurzbeschreibung:
Die Dialogbox „Allgemeine Einstellungen“ wird aufgerufen. Benutzerangaben werden festgelegt, verworfen, überschrieben und schließlich durch Beendigung des XCTL-Programmes gesichert. Das XCTL-Programm wird erneut gestartet und das Fenster „Allgemeine Einstellungen“ aufgerufen, um die aktuellen Benutzerangaben zu vergleichen.

Anwendungsfall „Halbwertsbreite messen“

Aufgabe

Diese Testfallsammlung überprüft die Funktionalitäten zur Messung der Halbwertsbreite (HWB). Die HWB wird durch Ausführen des Makros „InquireHwb“ aus der Datei `STANDARD.MAK` berechnet, weshalb mit einem Test der Halbwertsbreiten-Funktionalität hauptsächlich die Makrosprachen-Funktionalität überprüft wird. Makro-Ausführung ist eigentlich Gegenstand des Testpaketes zur „Ablaufsteuerung“. Getestet werden soll hierbei jedoch die Möglichkeit, aus der Dialogbox „Manuellen Justage“ heraus eine Messung der Halbwertsbreite anzustoßen.

Bemerkungen

Zur Bestimmung der Halbwertsbreite wird zunächst linksläufig nach der Peakposition gesucht, um von dort aus die eigentliche Messung durchzuführen. Um einen relativ konstanten Messwert bei wiederholten Messungen zu erhalten, muss der Vorgang möglichst genau auf einem eindeutigen Peak gestartet werden. Der Algorithmus zur Peaksuche fährt zunächst den Antrieb „DF“ nach links, weshalb man ihn vor dem Start der HWB-Messung am besten einen Schritt rechts neben ein Intensitätsmaximum positioniert. In der Datei `TESTDEV.DAT` sollte man sich davon überzeugen, daß alle Intensitätswerte vor und hinter dem ausgewählten Peak streng monoton fallen.

Der zuständige Algorithmus zur Bestimmung der Halbwertsbreite sieht eine Schrittweite von 0.4 Sekunden für den Antrieb „DF’ vor, welcher sich aus der Manuellen Justage heraus leider nicht mehr beeinflussen läßt. Die Auflösung der virtuellen Testprobe `TESTDEV.DAT` für den Antrieb „DF“ beträgt jedoch 1.0 Sekunden, wodurch es bei wiederholten Messungen zu abweichenden Ergebnissen kommt. Der zufällige Anteil des 0-dimensionalen Detektors beeinflußt dabei das Suchverhalten des Algorithmus’ so, dass mehr oder weniger Schritte zum Auffinden eines Peaks benötigt werden. Leicht abweichende HWB-Werte und Endstellungen der Motoren von den Erwartungswerten sind die Folge und durch entsprechende Toleranzen zu berücksichtigen.

Testfälle

Testfall 1
ID: HWB.1
Name: Messung der Halbwertsbreite aus der Manuellen Justage heraus
Kurzbeschreibung:
Die Dialogbox zur „Manuellen Justage“ wird aufgerufen. Die Halbwertsbreite wird für die aktuelle Position ermittelt.

3.3.4 Fehler

Beim Ableiten von Testfällen aus den Verhaltens-Spezifikation, sind neben den bekannten auch neue Fehler des XCTL-Systems aufgedeckt worden, die den Dokumenten im Web-Repository hinzugefügt wurden.

1. Ablaufsteuerung

Nr.	Datum	Status	Wer?	Beschreibung
3	18.07.2002 09.08.2002	1ST FIX	hanisch hanisch	**Charakterisierung:** Kein Unterschied bei Auswahl von Option „Ausführlicher Report“ **Ursache:** Variable `bSendReport` wird vor Ausführung eines Makros unabhängig vom Zustand der Checkbox immer auf `TRUE` gesetzt **Behebung:** `Steering.Visualising(TRUE, TRUE)` ersetzt durch `Steering.Visualising(Steering.IsResponding(), TRUE)`

2. Diffraktometrie/Reflektometrie, AreaScan

Nr.	Datum	Status	Wer?	Beschreibung
64	20.02.2002	1ST	hanisch	**Charakterisierung:** Dialog „PSD Kalibrierung“ lässt sich nur durch zweimaliges Anklicken des Buttons „Beenden“ schließen. **Ursache:** Die Ereignisse „Beenden gedrückt“ und „Eingabe eines Wertes für Winkel2Theta mit abschließendem Enter“, werden über die selbe Methode `TCalibratePsd::OnIDOK()` behandelt. Die Dialogbox schließt erst nach zweimaligen Aufruf dieser Methode, ohne zwischenzeitlicher Wertänderung bei „Winkel2Theta“. **Behebung:**

3. Motorsteuerung

Nr.	Datum	Status	Wer?	Beschreibung
14	15.01.2002	1ST	hanisch	**Charakterisierung:** Im Dialog „Grundstellung anfahren“ werden Optionen „Position beibehalten“ und „Grundstellung für alle Antriebe“ ausgegraut, wenn die Option „Gültige Kalibrierungsdaten“ deaktiviert wird. Trotzdem wird beim Referenzpunktlauf der Zustand der ausgegrauten Optionen berücksichtigt. **Ursache:** In `TCalibrateDlg::Dlg_OnCommand()` werden in Abhängigkeit von `id_ValidCalibrationData` die Elemente `id_HoldPosition` und `id_AllMotorsReset` nur ausgegraut. Die Zustände der Variablen `bHoldPosition` und `bAllMotors` bleiben davon unberührt und beeinflussen den Referenzpunktlauf. **Behebung:**

Nr.	Datum	Status	Wer?	Beschreibung
15	29.08.2002	1ST	hanisch	**Charakterisierung:** Beim Überschreiten des Endlage-Schalters (nur C-832 Antriebe) erscheinen mehrere Messageboxen „Motor-Steuerung“ mit einer Warnung. Es sollte eigentlich nur eine einzige Messagebox erscheinen. **Ursache:** In `CALLBACK TC_832::LimitWatch()` wird die Variable `MessageIsPosted` definiert, die wahrscheinlich für diesen Zweck vorgesehen wurde aber nirgends gesetzt oder ausgelesen wird. **Behebung:**

3.4 Die Ressourcen-Datenbank

Die Idee zur Entwicklung einer Datenbank zur Verwaltung von Fenstern, Dialogboxen, Menüs und Steuerelemente eines Testobjektes entstand aus der Erfahrung, dass sich die Benutzerschnittstellen im Zuge der Projektarbeit ständig verändern. Das geschieht sowohl grafisch (Dialogoberfläche), als auch programmintern (Ressourcen-IDs). Testskripte aus Regressionstestpaketen sollten so weit wie möglich von diesen Veränderungen unberührt bleiben.

Alle Kommandos der neu entwickelten Skriptsprache HTS sprechen Oberflächen-Elemente einer Anwendung über aussagekräftige Bezeichner an. Eine Datenbank soll dafür die konsistente Zuordnung zwischen Bezeichnern und den Oberflächen-Elementen verwalten. Dialogboxen und Fenster sind über ihren Titel (Überschrift) sehr gut identifizierbar, wenn sie innerhalb einer Anwendung eindeutig sind. Für Popup-Menüs und Menüpunkte sollten die vom Programm verwendeten Bezeichner hergenommen werden. Die Benennung von Steuerelementen gestaltet sich dagegen etwas schwieriger. Mehrere Buttons mit gleicher Beschriftung in einer Dialogbox müssen zur Ansteuerung voneinander unterscheidbar sein. Hier empfiehlt sich die Namenswahl unter Berücksichtigung der Aufgaben der Steuerelemente vorzunehmen. Ein identischer Bezeichner für mehrere Steuerelemente darf nur verwendet werden, wenn sich selbige in ihren Typen (Button, Scrollbar, Combobox etc.) unterscheiden.

In den Ressourcen-Dateien (`.RC`-Dateien) eines oberflächenbasierten Windows-Programmes sind nahezu all ihre Dialogboxen inclusive ihrer Steuerelemente vorzufinden, welche beim Übersetzen der Quellen in eine ausführbare Anwendung einbezogen werden. Die Zuordnung zwischen Control-IDs und ihren numerischen Repräsentanten wird durch `#define`-Anweisungen in entsprechenden Headerdateien vorgenommen. Diese Informationen machen wir uns zunutze, um einen großen Teil möglicher Oberflächenelemente automatisiert einzulesen und für die Ressourcen-Datenbank zu verwenden. Viele Dialog- und Steuerelementbezeichner werden im Zuge dieses Vorganges aus den Ressourcen eingelesen und genügen in den meisten Fällen als aussagekräftige Namen. Uneindeutige oder unqualifizierte Bezeichner können bei Bedarf mit dem *RCParser* modifiziert werden.

Das Auffinden von Oberflächen-Elemente aus den Ressourcen-Dateien wird mit Hilfe der Matcher-Komponente (siehe Kapitel 4 und Anhang A) und der Regeldatei `borland.rul` realisiert. In naher Zukunft soll das Projekt auf die Entwicklungsumgebung von Microsoft übertragen werden. Mit einer entsprechenden Regeldatei (`msvc.rul`), können syntaktische Unterschiede zwischen den Ressourcen-Dateien für die Entwicklungsumgebungen von Borland und Microsoft unproblematisch berücksichtigt werden.

Einige Dialogboxen oder Menüs sind in den `.RC`-Dateien nicht vorzufinden und müssen der Datenbasis manuell hinzugefügt werden. Für Standard-Dialogboxen, die beispielsweise zum Öffnen oder Schließen von Dateien aufgerufen werden, bietet der *RCParser* die Möglichkeit, sogenannte Template-Dialogboxen mit ihren typischen Steuerelementen hinzuzufügen. Fenster, die zur Darstellung von Kurvengrafiken beim AreaScan und LineScan eingesetzt werden, sind ebenfalls nicht aus den `.RC`-Dateien ablesbar und müssen der Datenbank manuell hinzugefügt werden. Die Menüstruktur des XCTL-Systems wird erst dynamisch zur Laufzeit aufgebaut. Eine Zuordnung zwischen den programminternen IDs und den Menüpunkten erfordert demnach die Analyse des Quellcodes, was beim Entwurf der Projektdatei `XCTL deutsch.rpp` für das XCTL-System vollständig vorgenommen wurde.

Der *RCParser* verwaltet die Pfade und Dateinamen der `.RC`- und `.h`-Dateien einer Ressourcen-Datenbank in seinen eigenen Projektdateien (Endung `.rpp`). Neben allen vorgefundenen Dialogboxen, Menüs und Steuerelementen werden Konflikte und der Modifizierungszustand jedes einzelnen Elements abgespeichert. Beim Interpretieren von Testskripten sollen nur konfliktfreie Elementbezeichner für die Übersetzung in numerische IDs herangezogen werden. Zusätzlich lassen sich Dialogboxen, Fenster, Menüs und Steuerelemente nach belieben aktivieren, deaktivieren und umbenennen. Die Testsuite *ATOS* greift dafür auf eine „abgespeckte“ Datenbasis (`.urf`-Datei) zu, welche nur aktivierte und konfliktfreie Oberflächen-Elemente beinhaltet. Mit einer Aktualisierungsoption im *RCParser* lassen sich die Oberflächen-Elemente des geöffneten Projektes mit den vorliegenden `.RC`- und `.h`-Dateien synchronisieren, um die Datenbank an eventuelle Veränderungen der Benutzerschnittstelle anzupassen.

Unter diesen Umständen wäre es möglich eine Datenbank mit englischsprachigen Bezeichnern in eine entsprechende `.urf`-Datei zu exportieren. Dazu müssten mit einigem Aufwand im RCParser-Projekt alle englischsprachigen Dialogboxen aktiviert und von Konflikten bereinigt werden. Zusätzlich wären die Menüstrukturen und Fensterbezeichnungen anzupassen. Mit einer solchen Ressourcen-Datenbank können dann beliebige Testsequenzen für die englischsprachige Programmversion entworfen werden. Da die Elemente der Benutzerschnittstelle nur mit anderen Bezeichnern angesprochen werden, sind am funktionellen Verhalten des Testobjekts jedoch keine Veränderungen zu erwarten.

Der Umgang mit dem *RCParser* wird im Benutzerleitfaden [20] detailliert beschrieben. An dieser Stelle soll die Abbildung 3.4 zur Veranschaulichung genügen.

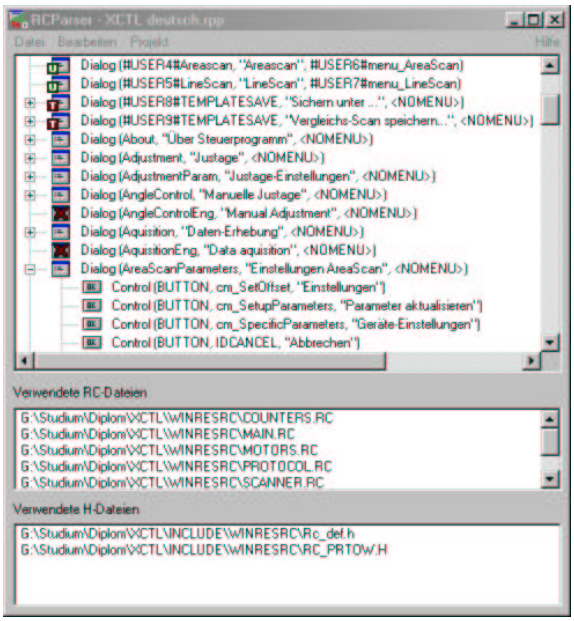


Abbildung 3.4: Einsatz des *RCParsers* am XCTL-System

Rudimente in den Ressourcen

Der *RCParser* kann außerdem einen Beitrag zur Bereinigung der Quellen des XCTL-Systems leisten, da alle vorgefundenen Dialogboxen und Menüs aus den `.RC`-Dateien übersichtlich zusammengestellt werden. Im Zuge der Entwicklung sind einige Ressourcen-Beschreibungen für Elemente, die es in aktuellen Programmversionen gar nicht mehr gibt, vergessen worden. Für jedes unbekannte Oberflächen-Element haben wir nach dessen Verwendung in den Projektquellen gesucht. Als Ergebnis entstand die Tabelle 3.3, in der rudimentäre Dialogboxen zusammengestellt werden und die für eine Bereinigung der .RC-Dateien dienen könnte.

Dialog-ID	Dialog-Titel	Ergebnis der Quellcode-Analyse
`CrossTableControl`	Kreuztisch: Objekt verfahren	Kein Quellcode für `TCrossTable`
`ExecuteCmd`	Terminal	Quellcode vorhanden, aber nirgendwo augerufen
`GetData`	Eingabe-Dialog	Quellcode vorhanden, aber nirgendwo augerufen
`ImageSaveOptions`	Speicher-Optionen	Kein Quellcode für Dialog vorhanden
`LineScanParameters`	Einstellungen LineScan	Kein Quellcode für Dialog vorhanden
`PSDRemoteSync`	PSD-Messung unter Fernsteuerung	Quellcode für Dialog unvollständig und auskommentiert
`PSDRemoteSyncEnd`	PSD-Measurements under remote Control	Quellcode für Dialog unvollständig und auskommentiert

Tabelle 3.3: Rudimentäre Dialogboxen des XCTL-Systems

Fazit

Unser kontextsensitives Testverfahren (siehe Abschnitt 2.4.1) spricht alle Steuerelemente der grafischen Benutzerschnittstelle (GUI) als logische Objekte an, wodurch die definierten Testskripte unabhängig von der Positionierung der Fenster, Dialogboxen und Steuerelemente sind. Die Verwaltung der Zuordnung zwischen programminternen ID's und evidenten Bezeichnern über eine externen Datenbank macht die definierten Testsequenzen des Regressionstests darüber hinaus noch resistenter gegenüber Veränderungen an der GUI der Testapplikation. Modifikationen an der Benutzerschnittstelle werden außerhalb der Testsequenzen mit dem *RCParser* erkannt und aktualisiert. Änderungen an den Kommandos in den Testskripten sind nur dann notwendig, wenn angesprochene Oberflächenelemente in einer aktuellen Programmversion nicht mehr existieren oder durch andere ersetzt wurden. Die Skripte sind sogar dann noch gültig, wenn die Testapplikation auf eine andere Entwicklungsumgebung portiert wird, solange die Beschreibung ihrer grafischen Benutzerschnittstelle in ähnlicher Weise über RC- und H-Dateien vorgenommen wird. Der Umgang mit syntaktischen Unterschieden ist Aufgabe des *RCParser*s und tangiert nicht die Testskripte.

3.5 Entwurf von Testskripten

Für den automatisierten Ablauf eines Regressionstestes haben wir bereits in unserer Studienarbeit [12] die Skriptsprache ATS (Atomic Testscript) entworfen. Einfache Kontrollkonstrukte, wenige Kommandos zur Fernsteuerung des Testobjektes und zur Zeitverwaltung, sowie ein paar Dateiverwaltungskommandos waren ausreichend, um Aktions- und Auswertungsschritte der spezifizierten Testfälle auszuführen.

Im Zuge der Arbeit entstand der Wunsch nach transparenteren Skriptkommandos. Aktivitäten wurden auf Nachrichten-Ebene über Control-IDs realisiert. Das erforderte tiefgreifende Kenntnisse über die Nachrichtenverwaltung von Windows und erschwerte die Konstruktion von Testsequenzen immens.

Um die Ergebnisse der Studienarbeit weiterhin nutzen zu können, entwickelten wir ein Verfahren zur Abbildung einer höheren auf die bereits vorhandene Skriptsprache. Dafür sollte der Quellcode des entwickelten Parsers weitestgehend unberührt bleiben. Wenn die Abbildung über Regeln in Textdateien realisiert wird, können sehr leicht beliebige Kommandos spezifiziert

werden und in eine Menge von Kommandos aus der tieferen Skriptsprache zur Ausführungszeit übersetzt werden.

Diese Idee motivierte uns zum Entwurf einer völlig neuen Skriptsprache HTS (Highlevel Testscripts), die zur Konstruktion zukünftiger Testskripte genutzt werden soll. Konzeptionell stellte sich die Verwendung von zwei Skriptebenen sogar als sehr praktisch heraus. Das Verhalten jedes Kommandos lässt sich leicht über die Regeldatei anpassen. Neue Kommandos lassen sich jederzeit dem Sprachschatz hinzufügen, solange sie sich in eine Menge von Kommandos der tieferen Skriptsprache ausdrücken lassen.

Wie sich jedoch herausstellen sollte, reichte der Vorrat an ATS-Kommandos für die Übersetzung der spezifizierten HTS-Kommandos nicht aus. Aus diesem Grund stellen wir im Abschnitt 3.5.2 noch einmal eine überarbeitete Version der Lowlevel-Skriptsprache ATS, zusammen. Neue bzw. überarbeitete Skriptkommandos werden dabei besonders hervorgehoben.

3.5.1 Von HTS nach ATS

Die Grundlage zur Realisierung bilden bereits entwickelte Komponenten wie LineParser und Matcher (siehe Kapitel 4). Im Mittelpunkt steht eine entsprechende Regeldatei (siehe Abbildung 3.5), die zur Übersetzung von Kommandos aus der Hochsprache HTS in Kommandos der tieferen Sprache ATS herangezogen wird. Der Parser muss sich dazu nur noch um die Konsistenz der Bezeichner für Menüs, Fenster, Dialogboxen und Steuerelemente und ihre Übersetzung in entsprechende Control-IDs kümmern. Dafür dient die in Kapitel 3.4 vorgestellte Ressourcen-Datenbank.

Jedes HTS-Kommando wird mit einer Vielzahl von ATS-Kommandos umschrieben. Mit diesem Verfahren ist es sehr leicht möglich, das Verhalten von HTS-Kommandos zu verändern und sogar neue Kommandos zu definieren. Dafür ist nur eine entsprechende Regel in der `PATTERN/OUTPUT`-Syntax des Matchers zu formulieren (siehe Anhang A). Der Umfang aller bekannten Steuerelement-Typen des `ACTION`-Kommandos könnte auf diese Weise problemlos erweitert werden. Ein Beispiel für die Übersetzung eines HTS-Skriptes in ein adäquates ATS-Skript ist im Anhang unter B.2 und B.3 vorzufinden.

Ein zu lösendes Problem stellte die Abbildung auftretender Fehler während der Ausführung von ATS-Kommandos in aussagekräftige Fehlerbeschreibungen für das aktuelle HTS-Kommando dar. Dem Tester nutzt die Aussage, dass `SEND` aufgrund eines ungültigen Zielfensters fehlschlug,

```
DESCRIPTION "HTS zu ATS"
BEGINSTATE  "NORMAL"                    # Anfangszustand des Matchers

######################################################################
#
# Aktionen (ACTION)
#
######################################################################

############ Menü
RULE
   STATE    "NORMAL"
   PATTERN  "ACTION",<SAVE:Window>,"MENU","CLICK",<SAVE:Menuitem>
   OUTPUT   "ISENABLED","MENU",<LOAD:Window>,<LOAD:Menuitem>
   OUTPUT   "ERROR","14","Das Fenster, in dem das Menü enthalten ist, konnte nicht gefunden werden !"
   OUTPUT   "ERROR","16","Der Menüpunkt ist nicht vorhanden !"
   OUTPUT   "ERROR","18","Der Menüpunkt ist nicht aktiv !"
   OUTPUT   "ERROR","19","Das Menü des Fensters konnte nicht gefunden werden !"
   OUTPUT   "POST",<LOAD:Window>,"NONE","273",<LOAD:Menuitem>,"0"
   OUTPUT   "POST",<LOAD:Window>,"NONE","15","0","0"
   NEWSTATE "NORMAL"
ENDRULE

############ Steuerelemente
###### Editbox
RULE
   STATE    "NORMAL"
   PATTERN  "ACTION",<SAVE:Window>,"EDITBOX","EDIT",<SAVE:Input>,<SAVE:Target>
   OUTPUT   "ISENABLED","CONTROL",<LOAD:Window>,<LOAD:Target>
   OUTPUT   "ERROR","14","Das Fenster, in dem die Editbox enthalten ist, konnte nicht gefunden werden !"
   OUTPUT   "ERROR","15","Die Editbox konnte nicht gefunden werden !"
   OUTPUT   "ERROR","18","Die Editbox ist nicht aktiv !"
   OUTPUT   "SETFOCUS",<LOAD:Window>,<LOAD:Target>
   OUTPUT   "ERROR","20","Auf die Editbox konnte kein Eingabefocus gesetzt werden !"
   OUTPUT   "SEND",<LOAD:Window>,<LOAD:Target>,"12","0",<LOAD:Input>
   NEWSTATE "NORMAL"
ENDRULE
...
```

Abbildung 3.5: Regeldatei `hts2ats.rul`

recht wenig. Es sollte eine Fehlermeldung im Kontext des momentan ausgeführten HTS-Kommandos ausgegeben werden. Genau für diese Zwecke wird das Schlüsselwort `ERROR` in der Regeldatei `hts2ats.rul` eingesetzt. Der ATS-Parser gibt bei Problemen während der Ausführung von ATS-Kommandos einen Fehlercode zurück, welcher sich im Kontext des aktuellen HTS-Kommandos in eine aussagekräftige Fehlermeldung formulieren lässt. Das erfordert selbstverständlich Kenntnisse über mögliche Rückkehrcodes des ATS-Parsers, die ausführlich in den Header- und Quelltextdateien dokumentiert sind.

3.5.2 Die Lowlevel-Skriptsprache ATS

BREAK

Aufruf: `BREAK`

Parameter: keine

Beschreibung: Dieses Kommando dient dazu, eine `LOOP`-Schleife sofort zu beenden. Die Schleife muss allerdings vom Typ `CYCLES` sein. Bei Ausführung dieses Kommandos wird direkt hinter das zugehörige `ENDLOOP`-Kommando gesprungen.

Beispiel:

```
LOOP,5,CYCLES,100
 ...
 BREAK  # An dieser Stelle wird die Schleife sofort
        # verlassen
 ...
ENDLOOP
  # An dieser Stelle wird das Skript fortgesetzt
```

CALC *neu!*

Dieses Kommando wurde neu eingeführt als sich herausstellte, dass zur Bestimmung einer API-Nachricht bei `SEND` oder `POST` oftmals Berechnungen notwendig sind.

Aufruf: `CALC,<VARIABLE>,<OPERATION>,<WERT>`

Parameter:

`<VARIABLE>`**:** Name der Variable mit deren Wert die Operation vorgenommen werden und die den resultierenden Wert empfangen soll. Wenn sie existiert, muss sie vom Typ `NUM` sein.

`<OPERATION>`**:** Hier sind `ADD` für die Addition und `SUB` für die Subtraktion zulässig.

`<WERT>`**:** Ein numerischer Wert, der zu dem Wert der Variablen addiert oder subtrahiert werden soll.

Beschreibung: Wenn die Variable bereits existiert, wird mit dem in `<WERT>` angegebenen Wert die geforderte Operation durchgeführt und das Ergebnis der Variable zugewiesen. Existiert die Variable noch nicht, so wird die Operation mit 0 und dem in `<WERT>` angegebenen Wert durchgeführt.

Beispiel:

```
CALC,var,ADD,1 # Zu dem Wert in der Variablen 'var' wird 1
               # addiert und wieder in 'var' gespeichert
CALC,var,SUB,5 # Von dem Wert in 'var' wird 5 subtrahiert
               # und in 'var' gespeichert
```

CLEANUP

Aufruf: `CLEANUP`

Parameter: keine

Beschreibung: Falls während der Ausführung eines ATS-Skriptes ein Fehler auftritt, wird direkt zu diesem Kommando gesprungen, sofern es vorhanden ist. Die mit `LAUNCH` aufgerufene Anwendung wird ggf. geschlossen. Alle folgenden Kommandos bis zum Ende des Skriptes dienen der Wiederherstellung des Zustandes vor Beginn der Ausführung des aktuellen Testskriptes. Temporär angelegte Dateien sollten hier gelöscht und originale Konfigurationsdateien wieder hergestellt werden, um das Testobjekt in seinen Ausgangszustand zu versetzen. Fehler beim Ausführen von Kommandos nach dem `CLEANUP`-Kommando werden ignoriert.

COMPARE *überarbeitet!*

Dieses Kommando wurde um den Parameter `<MODUS>` erweitert. Das macht diese Anweisung wesentlich flexibler, da nun auch beliebige Variablen als Sollwerte angegeben werden können.

Aufruf: `COMPARE,<ISTWERT>,<MODUS>,<SOLLWERT>[,<OPERATOR>]`

Parameter:

`<ISTWERT>`**:** Repräsentiert einen aktuellen Wert, der mit einem Sollwert verglichen wird. Es können folgende Bezeichner verwendet werden: `RESULT`, `WPARAM`, `LPARAM` oder der Bezeichner einer Variablen. Die Bezeichner `RESULT`, `WPARAM` und `LPARAM` stehen für Ergebnisse von speziellen Kommandos, z.B. `SEND`, `LAUNCHEXTERN`.

`<MODUS>`**:** Bestimmt ob ein textueller oder numerischer Vergleich zwischen Ist- und Sollwert durchgeführt werden soll. Zulässig sind hier `NUM` (numerisch) und `STR` (textuell).

`<SOLLWERT>`**:** Repräsentiert den Sollwert mit dem der Istwert verglichen wird. Hier gibt es mehrere Möglichkeiten. Zum einen kann man einen konstanten Wert z.B. `123` oder `"123"` angeben, zum anderen lassen sich auch Variablen als Sollwerte angeben.

`<OPERATOR>`**:** Dieser Parameter ist optional und wird nur bei numerischen Vergleichen beachtet. Wird er nicht angeben, so wird bei numerischen Vergleichen auf Gleichheit geprüft. Falls nicht auf Gleichheit geprüft werden soll, stehen folgende Operatoren zur Auswahl:

`GRT`**:** Istwert ist größer als Sollwert.

`GEQ`**:** Istwert ist größer-gleich Sollwert.

`TOL,<TOLERANZ>`: Istwert ist gleich Sollwert mit Toleranz, wobei `<TOLERANZ>` eine positive Konstante ist. D.h. es wird geprüft ob folgendes gilt

`<SOLLWERT>-<TOLERANZ>` $\leq$ `<ISTWERT>` $\leq$ `<SOLLWERT>+<TOLERANZ>`

`LEQ`: Istwert ist kleiner-gleich Sollwert.

`LSS`: Istwert ist kleiner als Sollwert.

`NEQ`: Istwert ist ungleich Sollwert.

Beschreibung: Dient zum Vergleich zweier Werte, z.B. von Ergebnissen mit Sollwerten. Wenn der Vergleich scheitert, wird der Testvorgang abgebrochen, außer wenn das `COMPARE`-Kommando die Bedingung für ein `IF`- oder `IFNOT`-Kommando ist. In diesem Falle wird das Ergebnis als "Wahr" oder "Falsch" ausgewertet und für die Verzweigung benutzt.

Beispiele:

```
COMPARE,RESULT,NUM,10          # Ueberpruefen ob 'RESULT' gleich 10
COMPARE,var,NUM,12,LSS         # Ueberpruefen ob Wert von 'var'
                               # kleiner 12
COMPARE,RESULT,NUM,var,TOL,2 # Ueberpruefen ob 'RESULT' innerhalb
                               # der Tolanzgrenzen '2' um den Wert
                               # von 'var' liegt
COMPARE,WPARAM,STR,"Hallo"   # Ueberpruefen ob 'WPARAM' dem String
                               # 'Hallo' entspricht
```

ELSE

Aufruf: `ELSE`

Parameter: keine

Beschreibung: Alle folgenden Kommandos bis zum zugehörigen `ENDLOOP` des aktuellen `IF`- bzw. `IFNOT`-Blockes dienen als Alternativzweig. Falls die Bedingung zu "Falsch" bzw. "Wahr" auswertet, werden alle Kommandos ab dem nächsten `ELSE` (falls vorhanden) innerhalb des Verzweigungsblockes ausgeführt. Ein `ELSE`-Kommando innerhalb eines Verzweigungsblockes ist optional.

Beispiel: siehe Kommando `IF`

ENDIF

Aufruf: `ENDIF`

Parameter: keine

Beschreibung: Schließt einen `IF`- oder `IFNOT`-Verzweigungsblock ab.

Beispiel: siehe Kommando `IF`

ENDLOOP

Aufruf: `ENDLOOP`

Parameter: keine

Beschreibung: Bildet das Ende eines `LOOP`-Schleifenblockes vom Typ `CYCLES`.

Beispiel: siehe Kommando `LOOP`

FILECOPY

Aufruf: `FILECOPY,<QUELLE>,<ZIEL>[,FORCE]`

Parameter:

`<QUELLE>`**:** Gibt den Pfad der Quelldatei relativ zum Arbeitsverzeichnis an. Es können auch Unterverzeichnisse angegeben werden. Zu beachten ist allerdings, dass für ein `\` die Zeichenfolge `\\` zu schreiben ist. Eine gültiger Quelldatei-Name wäre z.B. `"REF\\TEST01.REF"`. Der Name muß immer mit `"` eingeschlossen sein.

`<ZIEL>`**:** Hier gilt dasselbe, wie für den Parameter `<QUELLE>`.

`FORCE`**:** Diese Direktive ist optional und notwendig, wenn die Zieldatei bereits exisitert und entweder versteckt oder schreibgeschützt ist. Ohne diese Direktive scheitert das Kopieren in diesem Falle und verursacht einen Fehler. Die Direktive muß mit Vorsicht benutzt werden, da es somit möglich ist, Dateien zu manipulieren, die evtl. nicht manipuliert werden sollen!

Beschreibung: Kopiert eine Datei innerhalb des Arbeitsverzeichnisses.

Beispiele:

```
FILECOPY,"TEST","TEST.BAK"  # Kopiert die Datei 'TEST'
                            # in die Datei 'TEST.BAK'

FILECOPY,"TEST","READONLY",FORCE  # Kopiert die Datei
                                  # 'TEST' in die
                                  # Datei 'READONLY'
                                  # auch wenn letztere
                                  # schreibgeschuetzt
                                  # ist
```

FILEDELETE

Aufruf: `FILEDELETE,<DATEINAME>[,FORCE]`

Parameter:

`<DATEINAME>`**:** Gibt den Pfad der zu löschenden Datei relativ zum Arbeitsverzeichnis an. Es können auch Unterverzeichnisse angegeben werden. Zu beachten ist allerdings, dass für ein \ die Zeichenfolge \\ zu schreiben ist und dass der Name in " eingeschlossen sein muss (z.B. `"FILE.BAK"`).

`FORCE`**:** Ist eine optionale Direktive, die angegeben werden sollte, wenn die zu löschende Datei entweder versteckt oder schreibgeschützt ist. Die Direktive muss mit Vorsicht benutzt werden, da es somit möglich ist, Dateien zu löschen, die evtl. nicht gelöscht werden sollen!

Beschreibung: Löscht die angegebene Datei im Arbeitsverzeichnis. Falls die Datei nicht existiert, tritt ein Fehler auf.

Beispiele:

```
FILEDELETE,"DEL"  # Loescht die Datei 'DEL'

FILEDELETE,"DEL",FORCE  # Loescht die Datei 'DEL' auch wenn
                        # diese schreibgeschuetzt ist
```

FILEEXISTS

Aufruf: `FILEEXISTS,<DATEINAME>`

Parameter:

`<DATEINAME>`**:** Der Name der Datei, deren Existenz zu überprüfen ist. Der Dateiname kann auch Pfadangaben enthalten, wobei ein \ durch \\ zu ersetzen ist. Der Name bezeichnet eine Datei relativ zum Arbeitsverzeichnis. Der Dateiname muss von " eingeschlossen sein (z.B. `"RESULT.DAT"`).

Beschreibung: Prüft ob die angegebene Datei vorhanden ist. Wenn die Datei nicht vorhanden ist, tritt ein Fehler auf, außer das `FILEEXISTS`-Kommando ist eine Bedingung für ein `IF`- oder `IFNOT`-Kommando. In diesem Falle wird es zu "Wahr" oder "Falsch" ausgewertet.

Beispiel:

```
FILEEXISTS,"EXIST"  # Ueberpruefen ob die Datei 'EXIST'
                    # vorhanden ist
```

IF und IFNOT

Aufruf: `IF` oder `IFNOT`

Parameter: keine

Beschreibung: Beginnt einen Verzweigungsblock. Das folgende Kommando wird als Bedingung interpretiert. Wenn die Bedingung für `IF` zu "Wahr" auswertet, dann werden die Kommandos hinter der Bedingung bis zu einem `ELSE` oder bis zum Ende des Verzweigungsblockes ausgeführt. Wenn die Bedingung hingegen zu "Falsch" auswertet, dann werden die Kommandos hinter dem `ELSE` oder dem Ende des Verzweigungsblockes ausgeführt. Bei `IFNOT` verhält es sich umgekehrt. Als Bedingung sind die Kommandos `COMPARE`, `WAITFORWINDOW`, `ISENABLED` und `FILEEXISTS` zulässig.

Beispiele:

```
IF
  # Die Anweisung an dieser Stelle wird, wenn moeglich
  # als Bedingung ausgewertet

  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingung zu 'Wahr' auswertet
ENDIF

IF
  # Die Anweisung an dieser Stelle wird, wenn moeglich
  # als Bedingung ausgewertet

  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingung zu 'Wahr' auswertet
ELSE
  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingung zu 'Falsch' auswertet
ENDIF

IFNOT
  # Die Anweisung an dieser Stelle wird, wenn moeglich
  # als Bedingung ausgewertet

  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingung zu 'Falsch' auswertet
ELSE
  # Die Anweisungsfolge an dieser Stelle wird ausgefuehrt,
  # wenn die Bedingung zu 'Wahr' auswertet
ENDIF
```

INSERTLOG

Aufruf: `INSERTLOG,<DATEINAME>`

Parameter:

`<DATEINAME>`**:** Gibt den Namen der Datei relativ zum Arbeitsverzeichnis an, die in das Testprotokoll eingefügt werden soll. Der Name kann auch Pfadangaben enthalten. Allerding muss anstatt \ die Zeichenfolge \\ benutzt werden und der Name mus von " eingeschlossen sein (z.B. `"RESULT.LOG"`).

Beschreibung: Fügt eine (Text-)Datei in das Testprotokoll ein. Die angegebene Datei sollte im ASCII-Format vorliegen. Eine solche Datei könnte z.B. ein Ergebnisprotokoll eines externen Testprogrammes sein.

Beispiel:

```
INSERTLOG,"EXTLOG"  # Fuegt den Inhalt der Datei 'EXTLOG'
                    # in das aktuelle Testlog ein
```

ISENABLED *neu!*

Mit den vorhandenen Kommandos zur Übermittlung von Windows-Nachrichten über `POST` und `SEND` war es leider nicht möglich, den Zustand eines Steuerelements abzufragen. Einige Nachrichten geben als Rückgabewerte Pointer auf komplexe Strukturen zurück, auf die unsere Skriptsprache nicht ausgelegt ist. Ob ein Button „ausgegraut“ ist, ließ sich somit nur über die Definition eines völlig neuen Skriptkommandos realisieren. Auf diese Abfragemöglichkeit sollte nicht verzichtet werden, weil vor jeder Aktion mit einem Steuerelement sicher gestellt werden sollte, ob es momentan aktiv ist oder nicht.

Aufruf: `ISENABLED,<MODUS>,<FENSTER>,<CONTROL/MENU>`

Parameter:

`<MODUS>`**:** Wenn man ermitteln möchte, ob ein Fenster/Control aktiv ist, dann ist hier `CONTROL` anzugeben. Wenn man einen Menüeintrag überprüfen möchte, dann `MENU`.

`<FENSTER>`**:** Ist äquivalent zum Parameter im Kommando `SEND`.

`<CONTROL/MENU>`**:** Wenn in `<MODUS>` das Wort `CONTROL` angegeben wurde, dann ist dieser ebenfalls äquivalent zu dem Parameter im Kommando `SEND`. Wenn hingegen `MENU` als Modus vorliegt, dann muss hier die numerische ID des Menüeintrages spezifiziert werden.

Beschreibung: Ermittelt ob ein Fenster/Control bzw. ein Menüeintrag aktiv ist, d.h. ob es/er z.B. anklickbar ist. Das Ergebnis wird zusätzlich in `RESULT` gespeichert (1 für aktiv, 0 für inaktiv). Wenn dieser Befehl eine Bedingung ist, dann ist diese "Wahr", wenn das Element aktiv ist, sonst "Falsch". Wenn dieser Befehl keine Bedingung ist, tritt ein Fehler auf, falls das Element nicht aktiv ist.

Beispiel:

```
ISENABLED,CONTROL,"Test",2  # Prüft ob das Control mit der ID '2'
                            # im Fenster 'Test' aktiv ist

ISENABLED,MENU,"Test",123   # Prüft ob der Menüeintrag mit der ID
                            # '123' im Fenster 'Test' aktiv ist
```

KEY *neu!*

Eine Möglichkeit zur Simulation von Tastatur-Eingaben fehlte in der bisherigen Spezifikation gänzlich. Messungen können zum Beispiel mit ESC abgebrochen werden. Antriebe lassen sich im Schrittbetrieb mit der Tastatur bedienen.

Aufruf: `KEY,<TASTENCODE>,<AKTION>`

Parameter:

`<TASTENCODE>`**:** Gibt den numerischen Code der Taste an, die die Aktion ausführen soll. Zu beachten ist, dass es sich hier um die Codes der virtuellen Tasten (*virtual-key codes*) handelt. Diese sind aus der Dokumentation der Win32-API zu entnehmen.

`<AKTION>`**:** Dieser Parameter kann die Werte `PRESS` zum Drücken bzw. `RELEASE` zum Loslassen von Tasten annehmen. Es ist zu beachten, dass einige Tasten explizit losgelassen werden müssen, um eine Wirkung zu erzielen. Eine solche Taste ist z.B. die *Umschalt*-Taste.

Beschreibung: Simuliert eine Aktion, die von der Tastatur getätigt wird.

Beispiel:

```
KEY,65,PRESS    # "Drueckt" die A-Taste
KEY,65,RELEASE  # "Loest"die A-Taste
```

LAUNCH

Aufruf: `LAUNCH,<AUFRUF>,<FENSTERTITEL>`

Parameter:

`<AUFRUF>`**:** Gibt den Aufruf einer Anwendung relativ zum Arbeitsverzeichnis an. Der Aufruf muss von " umschlossen sein und anstatt \ ist \\ zu benutzen. Es können auch Optionen für den Start der Anwendung angegeben werden (z.B. `"PROGRAMM.EXE -i"`). Falls das Programm nicht gestartet werden konnte, tritt ein Fehler auf.

`<FENSTERTITEL>`**:** Beschreibt den Titel des Hauptfensters der zu startenden Anwendung. Der Titel muss von " eingeschlossen sein (z.B. `"Fenster"`). Falls das Fenster nach dem Start nicht erscheint, tritt ein Fehler auf.

Beschreibung: Mit diesem Kommando sollte die zu testende Anwendung gestartet werden.

Beispiel:

```
LAUNCH,"TEST.EXE -a -h","Titel"  # Startet das Programm
                                 # 'TEST.EXE' mit den
                                 # Optionen '-a -h' und
                                 # das Fenster mit dem
                                 # Titel 'Titel' wird
                                 # als Haupfenster
                                 # betrachtet
```

LAUNCHEXTERN

Aufruf: `LAUNCHEXTERN,<AUFRUF>,<TIMEOUT>`

Parameter:

`<AUFRUF>`**:** Gibt den Aufruf einer Anwendung relativ zum Arbeitsverzeichnis an. Der Aufruf muss von " umschlossen sein und anstatt \ ist \\ zu benutzen. Es können auch Optionen für den Start der Anwendung angegeben werden. (z.B. `"PROGRAMM.EXE -i"`). Falls das Programm nicht gestartet werden konnte, tritt ein Fehler auf.

`<TIMEOUT>`**:** Beschreibt die Zeitspanne, die auf das Beenden der gestarteten Anwendung gewartet werden soll. Die Zeitspanne wird in Millisekunden angegeben. Falls die Anwendung nach Ablauf der Zeitspanne nicht beendet ist, tritt ein Fehler auf. Mit `FOREVER` wird auf das Beenden unabhängig von der Laufzeit des gestarteten Programmes gewartet.

Dies sollte jedoch nur in Ausnahmefällen geschehen, weil die Beendigung bzw. ein Abbruch der Testdurchführung nicht garantiert werden kann. Mit `NOWAIT` kann der Parser angewiesen werden nicht auf das Terminieren zu warten und mit der Skriptausführung fortzusetzen.

Beschreibung: Es wird eine externe Anwendung gestartet. Solche Anwendungen könnten externe Testprogramme sein. Der Rückgabewert der Anwendung, wird in der Ergebnisvariablen `RESULT` gespeichert und kann mit `COMPARE` untersucht werden.

Beispiel:

```
LAUNCHEXTERN,"EXT.EXE -i",5000  # Startet das Programm
                                # 'EXT.EXE' mit der
                                # Option '-i' und wartet
                                # maximal 5000 ms auf
                                # dessen Beendigung
```

LOOP

Aufruf: `LOOP,<DAUER>,<TYP>,<VERZÖGERUNG>`

Parameter:

`<DAUER>`**:** Gibt die Gesamtdauer an, die die Schleife abgearbeitet werden soll. Bei Schleifen vom Typ `CYCLES` ist es die Anzahl der Zyklen, die die Schleife durchlaufen soll. Bei `TIME`-Schleifen hingegen wird die Gesamtdauer in Millisekunden angegeben und beschreibt wie lange die Schleife wiederholt werden soll.

`<TYP>`**:** Es gibt zwei Arten von Schleifen. Schleifen vom Typ `CYCLES` werden so oft durchlaufen, wie es im Parameter `<DAUER>` angegeben wurde. Dem `LOOP` dürfen dann mehrere Kommandos folgen, die durch ein `ENDLOOP` abgeschlossen werden. Alle Kommandos bis zum `ENDLOOP` werden bei jedem Zyklus wiederholt.

Bei `TIME`-Schleifen wird die Schleife solange durchlaufen, wie die in `<DAUER>` angegebene Zeitspanne noch nicht abgelaufen ist. Solchen Schleifen darf nur ein Kommando folgen ohne anschließendes `ENDLOOP`. Das folgende Kommando wird dann bei jedem Durchlauf wiederholt.

`<VERZÖGERUNG>`**:** Beschreibt die Zeit im Millisekunden, die nach jedem Zyklus gewartet werden soll. Diese Wartezeit geht bei `TIME`-Schleifen in die Berechnung der Gesamtdauer mit ein.

Beschreibung: Bildet den Anfang eines Schleifenblocks. Wenn in Schleifen vom Typ `CYCLES` ein `BREAK` ausgeführt wird, so ist die Schleife sofort beendet und das Ausführen der Kommandos wird hinter dem nächsten `ENDLOOP` fortgesetzt.

Beispiele:

```
LOOP,5,CYCLES,100
   # Fuehrt die hier eingefuegte Anweisungsfolge 5-mal
   # aus und wartet nach jedem Zyklus 100 ms
ENDLOOP

LOOP,5000,TIME,100
   # Fuehrt die hier eingefuegte Anweisung solange aus
   # bis 5000 ms abgelaufen sind und nach jedem Zyklus
   # wird 100 ms gewartet
```

MESSAGE

Aufruf: `MESSAGE,<MELDUNG>`

Paramater:

`<Meldung>`**:** Gibt die Meldung an, die dem Benutzer angezeigt werden soll. Sie muss von " umgeben sein (z.B. `"Das ist eine Meldung !"`).

Beschreibung: Erzeugt eine Meldung in Form einer Messagebox mit einem "OK" Button.

Beispiel:

```
MESSAGE,"Hello World"  # Erzeugt ein Nachrichtenfenster
                       # mit dem Text 'Hello World'
```

POST

Aufruf: `POST,<FENSTER>,<CONTROL>,<MESSAGE>,<WPARAM>,<LPARAM>`

Parameter: siehe Kommando `SEND`

Beschreibung: Dieses Kommando schickt eine Nachricht mittels der „Win32 API"-Methode `PostMessage()` an ein Fenster/Control. Allerdings wird hier kein Rückgabewert in der Ergebnisvariablen `RESULT` gespeichert. Sonst ist dieses Kommando äquivalent zum Kommando `SEND`.

Beispiele: siehe Kommando `SEND`

QUESTION

Aufruf: `QUESTION,<FRAGE>`

Parameter:

`<FRAGE>`**:** Die Meldung/Frage, die dem Benutzer angezeigt werden soll. Der Text muss in `"` eingeschlossen sein (z.B. `"Sind Sie sicher ?"`).

Beschreibung: Zeigt dem Benutzer eine Meldung/Frage in einer Messagebox mit einem "Ja"- und einem "Nein"-Button. In `RESULT` wird das Ergebnis in Abhängigkeit von der Antwort des Benutzers gespeichert. Den Wert 6, falls die Antwort "Ja" ist bzw. den Wert 7 bei "Nein".

Beispiel:

```
QUESTION,"Sind Sie sicher ?"  # Erzeugt ein Fenster,
                              # das die Frage
                              # 'Sind sie sicher ?'
                              # dem Benutzer zeigt
                              # und auf eine Antwort
                              # wartet
```

SAVE

Aufruf: `SAVE,<QUELLE>,<SPEICHER>,<MODUS>`

Parameter:

`<QUELLE>`**:** Gibt den Wert an, der zwischengespeichert werden soll. Zur Auswahl stehen `RESULT`, `WPARAM` und `LPARAM`.

`<SPEICHER>`**:** Beschreibt die Variable, in der der Wert gespeichert werden soll. Allerdings ist darauf zu achten, daß nicht spracheigene Bezeichner wie `WPARAM` oder numerische wie `1.0` verwendet werden.

`<MODUS>`**:** Gibt an, wie der Wert gespeichert werden soll. Es gibt die Modi `NUM` für numerisches und `STR` für textuelles Speichern. Die Art der Speicherung kann später den Vergleich mittels `COMPARE` beeinflussen. Siehe `COMPARE`.

Beschreibung: Speichert den spezifizierten Wert in einer Variablen und nimmt ggf. Konvertierungen zwischen textuellen und numerischen Formaten vor.

Beispiele:

```
SAVE,LPARAM,M1,NUM  # Wandelt den in 'LPARAM' gespeicherten
                    # String in eine Zahl und speichert
                    # diese in 'M1'

SAVE,RESULT,M2,STR  # Wandelt die in 'RESULT' gespeicherte
                    # Zahl in einen String und speichert
                    # diesen in 'M2'
```

SEND

Aufruf: `SEND,<FENSTER>,<CONTROL>,<MESSAGE>,<WPARAM>,<LPARAM>`

Parameter:

`<FENSTER>`**:** Spezifiziert das Zielfenster oder dessen Control an das die Nachricht geschickt werden soll. Wenn `MAIN` angegeben wird, ist das Ziel das Hauptfenster, welches durch `LAUNCH` ermittelt wurde. Es kann aber auch ein in " eingeschlossener Titel angegeben werden. Dann wird das Fenster als Ziel genommen, in dessen Titel die angegebene Zeichenfolge vorkommt. Es ist also darauf zu achten, dass der angegebene Titel für das gesamten System eindeutig ist. Beispielsweise kann für das Fenster mit dem Titel `"Fenster"` sowohl `"Fenster"`, als auch nur `"Fenst"` angegeben werden, sofern die Eindeutigkeit innerhalb der Testapplikation bewahrt bleibt.

`<CONTROL>`**:** Gibt evtl. das Control an, an das die Nachricht geschickt werden soll. Wenn hier `NONE` angegeben wird, geht die Nachricht direkt an das in `<FENSTER>` spezifizierte Fenster. Wenn das Ziel ein Control ist, muss hier die numerische ID des Controls stehen (z.B. `123`).

`<MESSAGE>`**:** Ist der numerische Wert der Nachricht, die geschickt werden soll und entspricht der „Win32 API"-Spezifikation für Windows-Nachrichten.

`<WPARAM>` **und** `<LPARAM>`**:** Deren Werte hängen von der in `<MESSAGE>` spezifizierten Nachricht ab und werden von der „Win32 API" vorgegeben. Bei bestimmten Nachrichten, verlangt die „Win32 API" dass evtl. ein Zeiger auf einen String (`LPSTR`) übergeben werden soll. In einem solchen Fall genügt es, die Zeichenkette `""` anzugeben. Nach der Ausführung von `SEND` befindet sich der Ergebnisstring in der Ergebnisvariablen `WPARAM` bzw. `LPARAM`. Es kann aber auch ein Name einer bereits existierenden Variable angegeben werden. Je nach Typ der Variablen wird der Wert entsprechend interpretiert.

Beschreibung: Sendet eine „Win32 API“-Nachricht an ein Fenster/Control. Dazu wird die Methode `SendMessage()` verwendet. Der Rückgabewert dieser Methode wird in der Ergebnisvariablen `RESULT` gespeichert. Manche Nachrichten dürfen nicht mittels `SendMessage()` versendet werden. In einem solchen Falle ist das Kommando `POST` zu verwenden.

Beispiele:

```
SEND,MAIN,NONE,273,0,0  # Sendet die Nachricht '273' an
                        # an das Hauptfenster mit den
                        # Parametern '0,0'

SEND,MAIN,1033,16,4,0  # Sendet die Nachricht '16' an das
                       # Control '1033' im Hauptfenster
                       # mit den parametern '4,0'

SEND,"Dialog",678,13,260,"" # Sendet die Nachricht '13'
                            # an das Control '678' im
                            # Fenster mit dem Titel
                            # 'Dialog' mit den Parametern
                            # '260' und einem Puffer fuer
                            # eine Zeichenkette, die
                            # in 'LPARAM' gespeichert wird
```

SETFOCUS *neu!*

Obwohl das Setzen des Eingabefokus' auch mit `SEND`- und `POST`-Kommandos möglich ist, sollte die Benutzung einer speziellen API-Methode vorgezogen werden, was mit der Anweisung `SETFOCUS` möglich ist.

Aufruf: `SETFOCUS,<FENSTER>,<CONTROL>`

Parameter: Die beiden Parameter sind äquivalent zu denen des Kommandos `SEND`.

Beschreibung: Setzt den Eingabefokus auf ein Fenster/Control.

Beispiel:

```
SETFOCUS,"Test",2  # Setzt dein Eingabefokus auf das Control
                   # mit der ID '2' im Fenster 'Test'
```

TESTFILECOPY

Aufruf: `TESTFILECOPY,<QUELLE>,<ZIEL>[,FORCE]`

Parameter:

`<QUELLE>`**:** Gibt den Pfad der Quelldatei relativ zum Testverzeichnisbaum an. Es können auch Unterverzeichnisse angegeben werden. Zu beachten ist allerdings, dass für ein \ die Zeichenfolge \\ zu schreiben ist. Eine gültige Quelldatei-Name wäre z.B. `"REF\\TEST01.REF"`. Der Name muss immer mit " eingeschlossen sein.

`<ZIEL>`**:** Hier gilt dasselbe, wie für den Parameter `<QUELLE>`. Allerdings bezieht sich der Name hierbei auf eine Datei relativ zum Arbeitsverzeichnis.

`FORCE`**:** Diese Direktive ist optional und ist notwendig, wenn die Zieldatei bereits exisitert und entweder versteckt oder schreibgeschützt ist. Ohne `FORCE` scheitert das Kopieren in diesem Falle und verursacht einen Fehler. Die Direktive muss mit Vorsicht benutzt werden, da es somit möglich ist, Dateien zu manipulieren, die evtl. nicht manipuliert werden sollen!

Beschreibung: Kopiert eine Datei aus den Testverzeichnisbaum in das Arbeitsverzeichnis.

Beispiele:

```
TESTFILECOPY,"TEST.REF","TEST.DAT"  # Kopiert die Datei
                                    # 'TEST.REF' aus dem
                                    # TVZB als 'TEST.DAT'
                                    # in das Arbeits-
                                    # verzeichnis

TESTFILECOPY,"SRC","READ",FORCE  # Kopiert die Datei
                                 # 'SRC' aus dem TVZB
                                 # als 'READ' in das
                                 # Arbeitsverzeichnis
                                 # auch wenn letztere
                                 # existiert und schreib-
                                 # geschuetzt ist
```

WAIT

Aufruf: `WAIT,<TIMEOUT>`

Parameter:

`<TIMEOUT>`**:** Gibt die Zeitdauer in Millisekunden an, die gewartet werden soll.

Beschreibung: Wartet die in `<TIMEOUT>` angegebene Zeit bevor mit der Ausführung des Testskriptes fortgefahren wird.

Beispiel:

```
WAIT,1500  # Wartet 1,5 Sekunden
```

WAITFORWINDOW

Aufruf: `WAITFORWINDOW,<FENSTERTITEL>,<TIMEOUT>`

Parameter:

`<FENSTERTITEL>`**:** Gibt den Titel des Fensters an, auf das gewartet werden soll. Der Titel muss in " eingeschlossen sein. Wenn der Titel des Fensters, das erscheinen soll z.B. `"Fenster"` ist, kann man entweder `"Fenster"` oder auch nur `"Fenst"` angeben, sofern diese Angabe innerhalb der Testapplikation eindeutig ist.

`<TIMEOUT>`**:** Gibt die Zeitspanne in Millisekunden an, die auf das Erscheinen des Fensters maximal gewartet werden soll. Bei `FOREVER` wird auf das Erscheinen des Fensters nach beliebiger Zeit gewartet. Dies sollte jedoch nur in Ausnahmefällen geschehen, weil die Beendigung bzw. ein Abbruch der Testdurchführung nicht garantiert werden kann.

Beschreibung: Wartet die in `<TIMEOUT>` angegebene Zeitspanne auf das Fenster mit dem Titel `<FENSTERTITEL>`. Wenn das Fenster nach der Zeitspanne nicht erschienen ist, tritt ein Fehler auf. Dieses Kommando kann auch als Bedingung für ein `IF` oder `IFNOT` fungieren. Es wertet zu "Wahr" aus, wenn das Fenster innerhalb der angegebenen Zeitspanne erscheint, sonst wertet es zu "Falsch" aus.

Beispiel:

```
WAITFORWINDOW,"Fenster",5000  # Wartet fuer 5 Sekunden auf
                              # das Fenster mit dem Titel
                              # "Fenster"
```

3.5.3 Die Highlevel-Skriptsprache HTS

Die HTS-Syntax

Hier wird die Grammatik der entwickelten Hochsprache in EBNF (erweiterte Backus-Naur-Form) vorgestellt. Bezeichner, die in " gesetzt sind, können aus beliebigen alphanumerischen Zeichen bestehen. Jede Anweisung wird mit einem Zeilenumbruch (\n) abgeschlossen. Folgende Morpheme werden vereinbart und besonders gekennzeichnet:

`<<zahl>>`	`zahl` ist eine beliebige natürliche Zahl, einschließlich 0
`<#zahl#>`	`zahl` ist eine beliebige natürliche Zahl, > 0
`<float>`	`float` ist eine beliebige Fließkommazahl
`$variable$`	`variable` ist eine beliebige Zeichenkette mit kleinen Buchstaben; Umlaute und Ziffern sind erlaubt

```
skript    = {anweisung}
            CLEANUP
            {anweisung}

anweisung = aktionsschritt | auswertungsschritt | interaktion |
            dateioperation | loopstruktur \n

aktionsschritt     = warten | messagebox_behandeln | tastaturoperation |
                     aktion
auswertungsschritt = lesen | vergleich | testen | fenstersichtbarkeit
interaktion        = frage | nachricht
dateioperation     = kopieren | löschen | exitenz | starten
loopstruktur       = LOOP, <#Wiederholungsanzahl#>
                     {anweisung}
                     ENDLOOP


Warten für eine Zeitspanne
--------------------------
warten    = WAIT, <#Millisekunden#>


Messageboxen behandeln
----------------------
messagebox_behandeln = HANDLEMESSAGEBOX, "Fenstertitel", OK|YES|NO|EXISTS


Tastaturoperationen
-------------------
tastaturoperation    = KEYBOARD, tastencode, <Drückdauer>
tastencode           = ESC | TAB | ENTER | LEFT | RIGHT | UP | DOWN
```

```
Aktionen auf der Oberfläche
---------------------------
aktion = fensteraktion | menueaktion | controlaktion

fensteraktion    = ACTION, MAIN|"Fenstertitel", WINDOW, fenstercode
fenstercode      = CLOSE |
                   MAXIMIZE |
                   MINIMIZE |
                   NORMALIZE

menueaktion      = ACTION, MAIN|"Fenstertitel", MENU, menueaktionscode,
                   "Menüpunkt" {,"Untermenüpunkt"}
menueaktionscode = CLICK

controlaktion    = ACTION, MAIN|"Fenstertitel", controltypaktion,
                   "Controlname"
controltypaktion = EDITBOX, EDIT, "Eingabe" |
                   EDITBOX, SET, "Eingabe" |
                   CHECKBOX, CHECK |
                   CHECKBOX, UNCHECK |
                   BUTTON, CLICK |
                   BUTTON, SELECT, "Buttonbeschriftung", <<max.Klickanz.>> |
                   HSCROLLBAR, LEFT, <<Klickdauer>> |
                   HSCROLLBAR, RIGHT, <<Klickdauer>> |
                   VSCROLLBAR, UP, <<Klickdauer>> |
                   VSCROLLBAR, DOWN, <<Klickdauer>> |
                   RADIOBUTTON, CHECK |
                   COMBOBOX, SELECT, "Auswahl" |
                   COMBOBOX, EDIT, "Eingabe" |
                   LISTBOX, SELECT, <#Zeilennummer#>
                   LISTBOX, SELECTLAST
```

```
Abfragen der Öberfläche/Elemente
--------------------------------
lesen = menuezustand | controlzustand

menuezustand            = READ, MAIN|"Fenstertitel", MENU,
                          menuezustandcode_lesen, $Variable$, "Menüpunkt"
                          {,"Untermenüpunkt"}
menuezustandcode_lesen  = ENABLESTATE

controlzustand          = READ, MAIN|"Fenstertitel", controltypzustand_lesen,
                          $Variable$, "Controlname"
```

```
controltypzustand_lesen = EDITBOX, TEXT |
                          EDITBOX, NUM |
                          EDITBOX, ENABLESTATE |
                          STATIC, TEXT |
                          STATIC, NUM |
                          RADIOBUTTON, CHECKSTATE |
                          RADIOBUTTON, ENABLESTATE |
                          CHECKBOX, CHECKSTATE |
                          CHECKBOX, ENABLESTATE |
                          BUTTON, TEXT |
                          BUTTON, ENABLESTATE |
                          HSCROLLBAR, POSITION |
                          HSCROLLBAR, ENABLESTATE |
                          VSCROLLBAR, POSITION |
                          VSCROLLBAR, ENABLESTATE |
                          COMBOBOX, TEXT |
                          COMBOBOX, ENABLESTATE |
                          LISTBOX, TEXT, <#Zeilennummer#> |
                          LISTBOX, TEXTLAST |
                          LISTBOX, ENABLESTATE


Abfragen und Vergleich der Überfläche/Elemente
----------------------------------------------
testen = menuetesten | controltesten

menuetesten              = TEST, MAIN|"Fenstertitel", MENU,
                           menuezustandcode_testen, "Menüpunkt"
                           {,"Untermenüpunkt"}
menuezustandcode_testen  = ENABLED |
                           DISABLED

controltesten            = TEST, MAIN|"Fenstertitel", controltypzustand_testen,
                           "Controlname"
controltypzustand_testen = EDITBOX, TEXT, "Editboxtext" |
                           EDITBOX, NUM, "Editboxwert" [,vergleichsmodus] |
                           EDITBOX, ENABLED |
                           EDITBOX, DISABLED |
                           STATIC, TEXT, "Statictext" |
                           STATIC, NUM, "Staticwert" [,vergleichsmodus] |
                           RADIOBUTTON, CHECKED |
                           RADIOBUTTON, UNCHECKED |
                           RADIOBUTTON, ENABLED |
                           RADIOBUTTON, DISABLED |
                           CHECKBOX, CHECKED |
                           CHECKBOX, UNCHECKED |
                           CHECKBOX, ENABLED |
                           CHECKBOX, DISABLED |
                           BUTTON, ENABLED |
```

```
                              BUTTON, DISABLED |
                              BUTTON, TEXT, "Buttonbeschriftung" |
                              HSCROLLBAR, POSITION, $variable$
                                [,vergleichsmodus] |
                              HSCROLLBAR, ENABLED |
                              HSCROLLBAR, DISABLED |
                              VSCROLLBAR, POSITION, $variable$
                                [,vergleichsmodus] |
                              VSCROLLBAR, ENABLED |
                              VSCROLLBAR, DISABLED |
                              COMBOBOX, TEXT, "Auswahltext" |
                              COMBOBOX, ENABLED |
                              COMBOCOX, DISABLED |
                              LISTBOX, TEXT, <#Zeilennummer#>, "Eintragtext" |
                              LISTBOX, TEXTLAST, "Eintragtext" |
                              LISTBOX, ENABLED |
                              LISTBOX, DISABLED


Vergleich von Ist- und Sollwerten
---------------------------------
vergleich = vergleich_mit_variable | vergleich_mit_wert

vergleich_mit_variable   = vergleich_var_textuell | vergleich_var_numerisch
vergleich_mit_wert       = vergleich_wert_textuell | vergleich_wert_numerisch

vergleich_var_textuell   = COMPARE, $Variable1$, STR, VAR, $Variable2$
vergleich_var_numerisch  = COMPARE, $Variable1$, NUM, VAR, $Variable2$
                           [,vergleichsmodus]

vergleich_wert_textuell  = COMPARE, $Variable1$, STR, VAL, "Wert"
vergleich_wert_numerisch = COMPARE, $Variable1$, NUM, VAL, "Wert"
                           [,vergleichsmodus]

vergleichsmodus       = GRT | GEQ | LEQ | LSS | NEQ | toleranz
toleranz              = TOL, <Toleranzwert>


Fenster vorhanden/nicht vorhanden
---------------------------------
fenstersichtbarkeit = WINDOWEXISTS, "Fenstertitel", YES|NO


Nachrichten/Fragen an Nutzer
----------------------------
nachricht = MESSAGE, "Text"
frage     = QUESTION, "Text", YES|NO
```

```
Datei kopieren
--------------
kopieren = COPY, dir, "Quelldatei", dir, "Zieldatei" [,FORCE]
dir      = ABS | TGT | REF | BIN | ENV | LOG


Datei existiert
---------------
existenz = EXISTS, dir, "Dateiname"


Datei löschen
-------------
löschen  = DELETE, dir, "Dateiname" [,FORCE] [,IFEXISTS]


Starten einer Anwendung
-----------------------
starten          = testobjekt_start | extern_start
testobjekt_start = START, "Parameter"


Starten eines externen Programmes
---------------------------------
extern_start           = start_warten_auf_ende | start_warten_auf_zeit |
                         start_ohne_warten
start_warten_auf_ende  = LAUNCH, dir, "Programmname", "Parameter",
                         FOREVER, [-]<#Returncode#>, [, "Logfile"]
start_warten_auf_zeit  = LAUNCH, dir, "Programmname", "Parameter",
                         TIME, <<Terminierungsdauer>>, [-]<#Returncode#>,
                         [, "Logfile"]
start_ohnewarten       = LAUNCH, dir, "Programmname", "Parameter", NOWAIT
```

Die Semantik der HTS-Kommandos

WAIT

Aufruf: `WAIT,<TIMEOUT>`

Parameter:

`<TIMEOUT>`**:** Gibt die Zeitdauer in Millisekunden an, die gewartet werden soll.

Beschreibung: Wartet die in `<TIMEOUT>` angegebene Zeit bevor mit der Ausführung des Testskriptes fortgefahren wird.

Beispiel:

```
WAIT,1500  # Wartet 1,5 Sekunden
```

HANDLEMESSAGEBOX

Aufruf: `HANDLEMESSAGEBOX, <TITEL>, <AKTION>, [<ANZAHL>]`

Parameter:

`<TITEL>`**:** Ist der Titel der Messagebox.

`<AKTION>`**:** Gibt an, mit welchem Aktion mit der Messagebox durchgeführt werden soll. Mögliche Werte sind:

`OK`	Anklicken des Buttons „Ok“
`YES`	Anklicken des Buttons „Ja“
`NO`	Anklicken des Buttons „Nein“
`EXISTS`	Prüft die Sichtbarkeit der Messagebox

`<ANZAHL>`**:** Dieser optionale Parameter gibt die maximale Anzahl der zu behandelnden Messageboxen an. In einigen Fällen können mehrere gleichnamige Messageboxen auf einmal auf dem Bildschirm erscheinen, was mit Angabe dieses Parameters berücksichtigt werden kann.

Beschreibung: Führt eine `<AKTION>` mit einer Messagebox `<TITEL>` durch.

Beispiel:

```
HANDLEMESSAGEBOX,"Meldung",OK    # Schließt Messagebox "Meldung"
                                 # mit Klick auf Button "Ok"

HANDLEMESSAGEBOX,"Fehler",YES,3  # Schließt maximal drei
                                 # Messageboxen mit dem Titel
                                 # "Fehler" durch Anklicken des
                                 # Buttons "Ja"
```

KEYBOARD

Aufruf: `KEYBOARD, <TASTE>, <DAUER>`

Parameter:

`<TASTE>`**:** Ist die zu aktivierende Taste. Mögliche Werte sind (derzeit) `ESC`, `TAB`, `ENTER`, `LEFT`, `RIGHT`, `UP`, `DOWN`.

`<DAUER>`**:** Gibt die Zeitdauer (auch 0 möglich), zwischen Drücken und Loslassen der Taste an.

Beschreibung: Simuliert einen Tastendruck auf der Tastatur für die Länge einer bestimmten Zeitdauer.

Beispiel:

```
KEYBOARD,ESC,0  # Drücken und sofortiges Loslassen
                # der ESC-Taste
```

ACTION

1. Fenster

Aufruf: `ACTION, <ZIEL>, WINDOW, <AKTION>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<AKTION>` bezieht. Kann auch `MAIN` für das Hauptfenster sein.

`<AKTION>`**:** Ist die durchzuführende Aktion. Mögliche Werte sind (derzeit) `CLOSE` zum Schließen, `MAXIMIZE` zum Maximieren der Fenstergröße, `MINIMIZE` zum Minimieren der Fenstergröße und `NORMALIZE` zum Zurücksetzen der ursprünglichen Fenstergröße.

Beschreibung: Fenster können geschlossen, maximiert und minimiert werden.

Beispiel:

```
ACTION,"LineScan",WINDOW,CLOSE  # Schließt das LineScan-Fenster
```

2. Menüs

Aufruf: `ACTION, <ZIEL>, MENU, <AKTION>, <MENÜPUNKT>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<AKTION>` bezieht. Kann auch `MAIN` für das Hauptfenster sein.

`<AKTION>`**:** Ist die durchzuführende Aktion. Mögliche Werte sind (momentan) `CLICK` zum Anklicken eines Menüpunktes. Geplant ist `CHECK` und `UNCHECK` zum Aktivieren (Abhaken) eines Menüpunktes, was mit den Mitteln der tieferen Skriptsprache ATS leider nicht möglich ist.

`<MENÜPUNKT>`**:** Eine beliebig lange Liste aus Popup-Menüs und einem Menüpunkt am Ende der Liste.

Beschreibung: Auswählen von (Kontext-)Menüpunkten in Fenstern und Dialogboxen.

Beispiel:

```
ACTION,MAIN,MENU,CLICK,"Öffnen","LineScan-Fenster"
  # Öffnet ein LineScan-Fenster aus dem Menü
  # des Hauptprogrammes

ACTION,"LineScan",MENU,CLICK,"Setup StepScan..."
  # Ruft aus dem Kontextmenü von "LineScan" die Dialogbox
  # "Einstellungen StepScan" auf
```

3. Controls

Aufruf: `ACTION, <ZIEL>, <CONTROLTYP>, <CONTROLAKTION>, <CONTROLNAME>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<CONTROLAKTION>` bezieht. Kann auch `MAIN` für das Hauptfenster sein.

`<CONTROLTYP>`**:** Ist ein Steuerelement-Typ aus folgenden möglichen Werten: `EDITBOX`, `CHECKBOX`, `BUTTON`, `RADIOBUTTON`, `HSCROLLBAR`, `VSCROLLBAR`, `COMBOBOX` und `LISTBOX`.

`<CONTROLAKTION>`**:** Ist die durchzuführende Aktion mit einem `<CONTROLTYP>`. Folgende Werte sind in Abhängigkeit des `<CONTROLTYP>`'s möglich:

`EDITBOX, EDIT, "Text"`	Setzt den Wert einer Editbox auf `"Text"`
`EDITBOX, SET, "Text"`	Setzt den Wert einer Editbox auf `"Text"` und schließt die Eingabe mit ENTER ab
`CHECKBOX, CHECK`	Aktiviert eine Checkbox
`CHECKBOX, UNCHECK`	Deaktiviert eine Checkbox
`BUTTON, CLICK`	Anklicken eines Buttons
`RADIOBUTTON, CHECK`	Aktiviert einen Radiobutton
`HSCROLLBAR, LEFT, <Dauer>`	Scrollbar nach links bewegen durch Anklicken des linken Endeelements für die mit `<Dauer>` angegebene Zeitlänge
`HSCROLLBAR, RIGHT, <Dauer>`	Scrollbar nach rechts bewegen durch Anklicken des rechten Endeelements für die mit `<Dauer>` angegebene Zeitlänge
`VSCROLLBAR, UP, <Dauer>`	Scrollbar nach oben bewegen durch Anklicken des oberen Endeelements für die mit `<Dauer>` angegebene Zeitlänge
`VSCROLLBAR, DOWN, <Dauer>`	Scrollbar nach unten bewegen durch Anklicken des unteren Endeelements für die mit `<Dauer>` angegebene Zeitlänge
`COMBOBOX, SELECT, "Text"`	Wählt `"Text"` aus einer Combobox (Achtung: evt. Leerzeichen beachten ! Bsp. `SELECT,"  reale Meßzeit"`)
`COMBOBOX, EDIT, "Text"`	Setzt `"Text"` in das Auswahlfeld einer Combobox (nicht möglich in XCTL, da alter Combobox-Typ)
`LISTBOX, SELECT, <Zeile>`	Wählt eine `<Zeile>` aus einer Listbox
`LISTBOX, SELECTLAST`	Wählt die letzte Zeile aus einer Listbox

`<CONTROLNAME>`**:** Ist der Bezeichner eines Steuerelements.

Beschreibung: Benutzung von wesentlichen Steuerelementen in Dialogboxen.

Beispiel:

```
ACTION,"Einstellungen StepScan",EDITBOX,EDIT,"-10.0","Minimum"
  # In die Editbox "Minimum" der Dialogbox mit dem Titel
  # "Einstellungen StepScan" wird der Wert "-10.0" eingetragen

ACTION,"Einstellungen",COMBOBOX,SELECT,"Counter","Detektorauswahl"
  # In der Combobox "Detektorauswahl" der Dialogbox mit dem Titel
  # "Einstellungen" wird der Eintrag "Counter" ausgewählt

ACTION,"Manuelle Justage",BUTTON,CLICK,"Verlassen"
  # Der Button "Verlassen" der Dialogbox mit dem Titel
  # "Manuelle Justage" wird angeklickt
```

READ

1. Menüs

Aufruf: `READ, <ZIEL>, MENU, <AKTION>, <VARIABLE>, <MENÜPUNKT>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<AKTION>` bezieht (kann auch `MAIN` für das Hauptfenster sein).

`<AKTION>`**:** Ist die durchzuführende Aktion. Der einzige mögliche Wert (momentan) ist `ENABLESTATE` zum Lesen des Zustandes, ob ein Menüpunkt auswählbar ist. Für Kontextmenüs ist diese Abfrage leider nicht möglich. Geplant ist `CHECKSTATE` zum Lesen des Zustandes, ob ein Menüpunkt ausgewählt (abgehakt) ist, was mit den Mitteln von ATS jedoch nicht möglich ist.

`<VARIABLE>` Ein Bezeichner (Kleinbuchstaben!) für eine Variable zur Aufnahme des abgefragten Zustandes.

`<MENÜPUNKT>`**:** Eine beliebig lange Liste aus Popup-Menüs und einem Menüpunkt am Ende der Liste.

Beschreibung: Abfrage und Zwischenspeichern von Zuständen eines Menüpunktes (auswählbar, abgehakt).

Beispiel:

```
READ,MAIN,MENU,ENABLESTATE,state,"Öffnen","LineScan-Fenster"
  # Schreibt in Variable "state" den Zustand, ob der Menüpunkt
  # "LineScan-Fenster" im Popup "Öffnen" auswählbar ist
  # oder nicht
```

2. Controls

Aufruf: `READ, <ZIEL>, <CONTROLTYP>, <CONTROLAKTION>,<VARIABLE>, <CONTROLNAME>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<CONTROLAKTION>` bezieht (kann auch `MAIN` für das Hauptfenster sein).

`<CONTROLTYP>`**:** Ist ein Steuerelement-Typ aus folgenden möglichen Werten: `EDITBOX`, `STATIC`, `CHECKBOX`, `BUTTON`, `RADIOBUTTON`, `HSCROLLBAR`, `VSCROLLBAR`, `COMBOBOX` und `LISTBOX`.

`<CONTROLAKTION>`: Ist die durchzuführende Aktion mit einem `<CONTROLTYP>`. Folgende Werte sind in Abhängigkeit des `<CONTROLTYP>`'s möglich:

`EDITBOX, TEXT`	Liest den Wert einer Editbox und interpretiert ihn als Zeichenkette
`EDITBOX, NUM`	Liest den Wert einer Editbox und interpretiert ihn als numerischen Wert
`EDITBOX, ENABLESTATE`	Liest den Zustand einer Editbox, ob sie Eingaben entgegen nehmen kann
`STATIC, TEXT`	Liest den Wert aus einem Static-Element und interpretiert ihn als Zeichenkette
`STATIC, NUM`	Liest den Wert aus einem Static-Element und interpretiert ihn als numerischen Wert
`RADIOBUTTON, CHECKSTATE`	Liest den Zustand eines Radiobuttons, ob er ausgewählt ist oder nicht
`RADIOBUTTON, ENABLESTATE`	Liest den Zustand eines Radiobuttons, ob er auswählbar ist oder nicht
`CHECKBOX, CHECKSTATE`	Liest den Zustand einer Checkbox, ob sie ausgewählt ist oder nicht
`CHECKBOX, ENABLESTATE`	Liest den Zustand einer Checkbox, ob sie auswählbar ist oder nicht
`BUTTON, TEXT`	Liest die Beschriftung eines Buttons (Achtung: `"&Verlassen"` für "Verlassen")
`BUTTON, ENABLESTATE`	Liest den Zustand eines Buttons, ob er anklickbar ist oder nicht
`HSCROLLBAR, POSITION`	Liest die relative Position einer horizontalen Scrollbar
`HSCROLLBAR, ENABLESTATE`	Liest den Zustand einer horizontalen Scrollbar, ob sie bewegt werden kann
`VSCROLLBAR, POSITION`	Liest die relative Position einer vertikalen Scrollbar
`VSCROLLBAR, ENABLESTATE`	Liest den Zustand einer vertikalen Scrollbar, ob sie bewegt werden kann
`COMBOBOX, TEXT`	Liest den momentan ausgewählten Eintrag einer Combobox
`COMBOBOX, ENABLESTATE`	Liest den Zustand einer Combobox, ob sie eine Auswahl zulässt
`LISTBOX, TEXT, <Zeile>`	Liest den Eintrag der `<Zeile>` aus einer Listbox
`LISTBOX, TEXTLAST`	Liest den letzten Eintrag aus einer Listbox
`LISTBOX, ENABLESTATE`	Liest den Zustand einer Listbox, ob sie eine Auswahl zulässt

`<CONTROLNAME>`: Ist der Bezeichner eines Steuerelements.

Beschreibung: Auslesen und Zwischenspeichern von Werten und Zuständen wesentlicher Steuerelemente einer Dialogbox.

Beispiel:

```
READ,"Manuelle Justage",HSCROLLBAR,POSITION,pos,"Scrollbar"
  # Die relative Position der horizontalen Scrollbar in
  # Dialogbox "Manuelle Justage" wird ausgelesen und in der
  # Variablen "pos" gespeichert.

READ,"Einstellungen",COMBOBOX,ENABLESTATE,state,"Detektorauswahl"
  # Der Zustand, ob die Combobox "Detektorauswahl" in der Dialogbox
  # "Einstellungen" momentan eine Auswahl zulässt oder nicht,
  # wird in der Variablen "state" abgespeichert.
```

TEST

1. Menüs

Aufruf: `TEST, <ZIEL>, MENU, <AKTION>, <MENÜPUNKT>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<AKTION>` bezieht. Kann auch `MAIN` für das Hauptfenster sein.

`<AKTION>`**:** Ist die durchzuführende Aktion. Mögliche Werte sind (derzeit) `ENABLED` und `DISABLED` zum Überprüfen, ob ein Menüpunkt auswählbar ist. Für Kontextmenüs ist diese Abfrage leider nicht möglich. Geplant ist `CHECKED` und `UNCHECKED` zum Überprüfen, ob ein Menüpunkt ausgewählt (abgehakt) ist, was mit den Mitteln von ATS jedoch nicht möglich ist.

`<MENÜPUNKT>`**:** Eine beliebig lange Liste aus Popup-Menüs und einem Menüpunkt am Ende der Liste.

Beschreibung: Überprüfung von Zuständen eines Menüpunktes (auswählbar, abgehakt).

Beispiel:

```
TEST,MAIN,MENU,ENABLED,"Öffnen","LineScan-Fenster"
  # Überprüft, ob der Menüpunkt "LineScan-Fenster"
  # im Popup "Öffnen" auswählbar ist oder nicht
```

2. Controls

Aufruf: `TEST, <ZIEL>, <CONTROLTYP>, <CONTROLAKTION>, <CONTROLNAME>`

Parameter:

`<ZIEL>`**:** Ist der Fenstertitel, auf den sich die `<CONTROLAKTION>` bezieht. Kann auch `MAIN` für das Hauptfenster sein.

`<CONTROLTYP>`**:** Ist ein Steuerelement-Typ aus folgenden möglichen Werten: `EDITBOX`, `STATIC`, `CHECKBOX`, `BUTTON`, `RADIOBUTTON`, `HSCROLLBAR`, `VSCROLLBAR`, `COMBOBOX` und `LISTBOX`.

`<CONTROLAKTION>`**:** Ist die durchzuführende Aktion mit einem `<CONTROLTYP>`. Folgende Werte sind in Abhängigkeit des `<CONTROLTYP>`'s möglich:

`EDITBOX, TEXT, "Text"`	Liest den Wert einer Editbox und vergleicht ihn als Zeichenkette mit `"Text"`
`EDITBOX, NUM, "Wert"[, <Modus>]`	Liest den Wert einer Editbox und vergleicht ihn numerisch mit `"Wert"`. Der optionale Parameter `<Modus>` bestimmt die Vergleichsmethode (`GRT`, `LSS` ...siehe `COMPARE`).
`EDITBOX, ENABLED`	Überprüft, ob Editbox Eingaben entgegen nehmen kann
`EDITBOX, DISABLED`	Überprüft, ob Editbox keine Eingaben entgegen nehmen kann
`STATIC, TEXT, "Text"`	Liest den Wert aus einem Static-Element und vergleicht ihn als Zeichenkette mit `"Text"`
`STATIC, NUM, "Wert"[, <Modus>]`	Liest den Wert aus einem Static-Element und vergleicht ihn numerisch mit `"Wert"`
`RADIOBUTTON, CHECKED`	Überprüft, ob Radiobutton ausgewählt ist
`RADIOBUTTON, UNCHECKED`	Überprüft, ob Radiobutton nicht ausgewählt ist
`RADIOBUTTON, ENABLED`	Überprüft, ob Radiobutton auswählbar ist
`RADIOBUTTON, DISABLED`	Überprüft, ob Radiobutton nicht auswählbar ist

`CHECKBOX, CHECKED`	Überprüft, ob Checkbox ausgewählt ist
`CHECKBOX, UNCHECKED`	Überprüft, ob Checkbox nicht ausgewählt ist
`CHECKBOX, ENABLED`	Überprüft, ob Checkbox auswählbar ist
`CHECKBOX, DISABLED`	Überprüft, ob Checkbox nicht auswählbar ist
`BUTTON, ENABLED`	Überprüft, ob Button anklickbar ist
`BUTTON, DISABLED`	Überprüft, ob Button nicht anklickbar ist
`BUTTON, TEXT, "Text"`	Überprüft, ob Button mit `"Text"` beschriftet ist (Achtung: `"&Verlassen"` für `"Verlassen"`)
`HSCROLLBAR, POSITION, <variable> [, <modus>]`	Vergleicht die relative Position einer horizontalen Scrollbar mit dem Wert aus `<variable>`. Der optionale Parameter `<Modus>` bestimmt die Vergleichsmethode (`GRT`, `LSS` ...siehe `COMPARE`).
`HSCROLLBAR, ENABLED`	Überprüft, ob horizontale Scrollbar bewegt werden kann
`HSCROLLBAR, ENABLED`	Überprüft, ob horizontale Scrollbar nicht bewegt werden kann
`VSCROLLBAR, POSITION, <variable> [, <modus>]`	Vergleicht die relative Position einer vertikalen Scrollbar mit dem Wert aus `<variable>`
`VSCROLLBAR, ENABLED`	Überprüft, ob vertikale Scrollbar bewegt werden kann
`VSCROLLBAR, ENABLED`	Überprüft, ob vertikale Scrollbar nicht bewegt werden kann
`COMBOBOX, TEXT, "Text"`	Vergleicht momentan eingestellten Eintrag einer Combobox mit `"Text"`
`COMBOBOX, ENABLED`	Überprüft, ob Auswahl in einer Combobox möglich ist
`COMBOBOX, DISABLED`	Überprüft, ob Auswahl in einer Combobox nicht möglich ist
`LISTBOX, TEXT, <Zeile>, "Text"`	Vergleicht die `<Zeile>` einer Listbox mit `"Text"`
`LISTBOX, TEXTLAST, "Text"`	Vergleicht die letzte Zeile einer Listbox mit `"Text"`
`LISTBOX, ENABLED`	Überprüft, ob Auswahl in einer Listbox möglich ist
`LISTBOX, DISABLED`	Überprüft, ob Auswahl in einer Listbox nicht möglich ist

`<CONTROLNAME>`: Ist der Bezeichner eines Steuerelements.

Beschreibung: Überprüfen von Werten und Zuständen wesentlicher Steuerelemente einer Dialogbox.

Beispiel:

```
TEST,"Manuelle Justage",EDITBOX,NUM,"0.0",TOL,0.1,"Neuer Winkel"
  # Überprüft, ob die Editbox mit der Bezeichnung "Neuer Winkel"
  # in der Dialogbox "Manuelle Justage" dem Zahlenwert 0.0 unter
  # Beachtung einer Toleranz von 0.1 entspricht

TEST,"Grundstellung anfahren",BUTTON,DISABLED,"Referenzpunktlauf"
  # Überprüft, ob der Button "Referenzpunktlauf" in der Dialogbox
  # "Grundstellung anfahren" deaktiviert (ausgegraut) ist
```

COMPARE

Aufruf: `COMPARE, <ISTWERT>, <NUM/STR>, <VAR/VAL>, <SOLLWERT> [,MODUS]`

Parameter:

`<ISTWERT>`: Ist ein Bezeichner für eine Variable, der den zu überprüfenden Istwert (Zahl oder Zeichenkette) enthält.

`<NUM/STR>`: Wenn Soll- und Istwert als numerische Werte interpretiert und verglichen werden, steht an dieser Stelle `NUM`. Für einen zeichenweisen Vergleich wird stattdessen `STR` verwendet.

`<VAR/VAL>`: Der Sollwert kann entweder eine Variable oder ein in "-Zeichen zu setzender Wert sein. Zur Unterscheidung beider Möglichkeiten wird an dieser Stelle entweder `VAR` oder `VAL` verwendet.

`<SOLLWERT>`: Ist in Abhängigkeit von `<VAR/VAL>` entweder ein Bezeichner für eine Variable, die den zu vergleichenden Sollwert (Zahl oder Zeichenkette) enthält, oder ein in "-Zeichen gesetzter Sollwert.

`<MODUS>`: Dieser optionale Parameter bestimmt bei einem numerischen Vergleich (`<NUM/STR>`=`NUM`) das Verfahren. Folgende Möglichkeiten werden unterstützt:

`GRT`	Istwert muss größer als der Sollwert sein
`GEQ`	Istwert muss größer-gleich dem Sollwert sein
`LSS`	Istwert muss kleiner als der Sollwert sein
`LEQ`	Istwert muss kleiner-gleich dem Sollwert sein
`NEQ`	Istwert und Sollwert müssen unterschiedlich sein
`TOL, <Tol.Wert>`	Istwert darf maximal vom Sollwert ±`<Tol.Wert>` abweichen

Beschreibung: Vergleicht Istwerte aus Variablen entweder direkt mit Sollwerten oder mit Werten aus anderen Variablen. Der Vergleich kann zeichenweise oder numerisch durchgeführt werden. Dieses Kommando arbeitet eng mit `READ` zusammen, da das Abspeichern von Werten in Variablen für einen späteren Vergleich erst durch `COMPARE` sinnvoll wird.

Beispiel:

```
COMPARE,startpos,NUM,VAR,endpos,LSS
  # Vergleicht numerisch den Wert der Variablen "startpos"
  # mit dem Wert der Variablen "endpos". Dabei muss
  # "startpos" kleiner als "endpos" sein, sonst wird mit
  # einer Fehlermeldung abgebrochen.
```

WINDOWEXISTS

Aufruf: `WINDOWEXISTS, <FENSTER>, <ABFRAGE>`

Parameter:

`<FENSTER>`: Ist der Titel des Fensters bzw. der Dialogbox, welches auf Existenz überprüft werden soll.

`<ABFRAGE>`: Soll überprüft werden, ob ein Fenster momentan existiert bzw. sichtbar ist, muss `YES` verwendet werden, andernfalls `NO`.

Beschreibung: Abfrage der Existenz bzw. Sichtbarkeit eines Fensters oder einer Dialogbox. Für Messageboxen wird eine Abfrage, ob die Dialogbox nach `HANDLEMESSAGEBOX` verschwunden ist, automatisch durchgeführt.

Beispiel:

```
WINDOWEXISTS,"LineScan",YES  # Überprüft, ob das Fenster
                             # "LineScan" momentan sichtbar ist
```

MESSAGE

Aufruf: `MESSAGE, <NACHRICHT>`

Parameter:

`<NACHRICHT>`**:** Ist ein in "-Zeichen eingeschlossener Text, der in der Messagebox dargestellt wird. Erlaubt sind C-typische Formatierungsanweisungen wie `\n` und `\t`.

Beschreibung: Während eines Testlaufs können Anweisungen an den Tester gegeben werden. Erst nach Bestätigung durch Anklicken des „Ok“-Buttons in der Messagebox wird mit der Testdurchführung fortgefahren.

Beispiel:

```
MESSAGE,"Bitte mit der Maus \n einmal auf die Grafik klicken!"
  # Fordert den Tester zum interaktiven Eingriff auf
```

QUESTION

Aufruf: `QUESTION, <FRAGE>, <ANTWORT>`

Parameter:

`<FRAGE>`**:** Ist ein in "-Zeichen eingeschlossener Text, der in der Messagebox dargestellt wird. Erlaubt sind C-typische Formatierungsanweisungen wie `\n` und `\t`.

`<ANTWORT>`**:** Mögliche Werte sind `YES` oder `NO` zur Festlegung der erwarteten Bestätigung mit „Ja“ oder „Nein“.

Beschreibung: Während eines Testlaufs können Fragen an den Tester gestellt werden. Erst nach Bestätigung durch Anklicken des „Ja“- oder „Nein“-Buttons in der Messagebox wird mit der Testdurchführung fortgefahren. Entspricht die Antwort des Testers nicht der mit `<ANTWORT>` festgelegten, wird die Durchführung des Skriptes an dieser Stelle abgebrochen.

Beispiel:

```
QUESTION,"Ist die Kurve in der Grafik verschwunden ?",YES
  # Fragt den Tester nach einem Ereignis und wartet auf
  # Bestätigung mit "Ja"; bei "Nein" wird abgebrochen
```

COPY

Aufruf: `COPY, <QUELLVERZEICHNIS>, <QUELLDATEI>, <ZIELVERZEICHNIS>, <ZIELDATEI> [,FORCE]`

Parameter:

`<QUELLVERZEICHNIS>`, `<ZIELVERZEICHNIS>`: Einige Pfade zu Projektverzeichnissen sind *ATOS* zur Ausführungszeit bekannt und können mit diesen Parametern angesprochen werden. Das ermöglicht die Portabilität von ATOS-Projekten (bzw. Testskripten), da sie weitestgehend unabhängig von absoluten Pfadangaben sind. Mögliche Werte sind:

`ABS`	in `<QUELLDATEI>` bzw. `<ZIELDATEI>` steht ein absoluter Pfad
`TGT`	Verzeichnis in der das Testobjekt liegt
`REF`	Verzeichnis mit Solldateien (\ref\)
`BIN`	Binary-Verzeichnis des Projektes (\bin\)
`ENV`	Verzeichnis der Umgebungsdateien des Projektes (\env\)
`LOG`	Verzeichnis zur Aufnahme der Logdateien (\log\)

`<QUELLDATEI>`, `<ZIELDATEI>`: Ist ein in "-Zeichen eingeschlossener Dateiname.

`FORCE`: Diese Direktive ist optional und notwendig, wenn die Zieldatei bereits existiert und versteckt oder schreibgeschützt ist. Ohne `FORCE` würde das Kopieren in diesem Falle scheitern und einen Fehler verursachen.

Beschreibung: Kopieren von Dateien aus Verzeichnissen eines ATOS-Projektes oder aus Verzeichnissen mit absoluten Pfaden.

Beispiel:

```
COPY,ENV,"ODIFF_HARDWARE.INI",TGT,"HARDWARE.INI",FORCE
  # Kopiert die Datei "\env\ODIFF_HARDWARE.INI" ins
  # Verzeichnis des Testobjekts, auch wenn die vorhandene
  # Datei schreibgeschützt ist
```

EXISTS

Aufruf: `EXISTS, <VERZEICHNIS>, <DATEI>`

Parameter:

`<VERZEICHNIS>`: Mögliche Werte sind: `ABS`, `TGT`, `REF`, `BIN`, `ENV` und `LOG` (siehe `COPY`)

`<DATEI>`: Ist ein in "-Zeichen eingeschlossener Dateiname.

Beschreibung: Überprüft die Existens einer Datei.

Beispiel:

```
EXISTS,TGT,"DEVELOP.INI"  # Überprüft, ob die Datei DEVELOP.INI
                          # im Verzeichnis des Testobjekts
                          # vorhanden ist.
```

DELETE

Aufruf: `DELETE, <VERZEICHNIS>, <DATEI> [,FORCE] [,IFEXISTS]`

Parameter:

`<VERZEICHNIS>`: Mögliche Werte sind: `ABS`, `TGT`, `REF`, `BIN`, `ENV` und `LOG` (siehe `COPY`)

`<DATEI>`: Ist ein in "-Zeichen eingeschlossener Dateiname.

`FORCE`: Diese Direktive ist optional und notwendig, wenn die Datei versteckt oder schreibgeschützt ist. Ohne `FORCE` würde das Löschen in diesem Falle scheitern und einen Fehler verursachen.

`IFEXISTS`: Diese Direktive ist optional und führt zu keinem Fehler bzw. Abbruch, wenn die zu löschende Datei nicht existiert.

Beschreibung: Löschen einer Datei.

Beispiel:

```
DELETE,ABS,"C:\\TEST1.CRV",IFEXISTS
  # Löscht die Datei C:\TEST1.CRV, wenn sie
  # vorhanden ist (evt. aus früheren Testläufen).

DELETE,TGT,"HARDWARE.BAK",FORCE
  # Löscht am Ende eines Testlaufs die
  # versteckte Datei HARDWARE.BAK aus
  # dem Verzeichnis des Testobjekts.
```

START

Aufruf: `START, <PARAMETER>`

Parameter:

`<PARAMETER>`**:** Sind in "-Zeichen eingeschlossene Parameter zur Übergabe an das zu startende Testobjekt.

Beschreibung: Startet die zu testende Anwendung (Testobjekt). Dateiname und Pfad sind der Testsuite *ATOS* bekannt.

Beispiel:

```
START,"-a -b"  # Startet das Testobjekt eines ATOS-Projekts
               # mit Übergabe von Parametern
```

LAUNCH

1. Warten auf Terminierung

Aufruf: `LAUNCH, <VERZEICHNIS>, <PROGRAMMNAME>, <PARAMETER>, FOREVER, <RETURNCODE>, [,<LOGFILE>]`

Parameter:

`<VERZEICHNIS>`**:** Mögliche Werte sind: `ABS`, `TGT`, `REF`, `BIN`, `ENV` und `LOG` (siehe `COPY`)

`<PROGRAMMNAME>`**:** Ist ein in "-Zeichen eingeschlossener Dateiname.

`<PARAMETER>`**:** Sind in "-Zeichen eingeschlossene Parameter zur Übergabe an das zu startende Programm.

`<RETURNCODE>`**:** Der Rückkehrcode des aufgerufenen Programmes wird mit `<RETURNCODE>` verglichen. Bei Übereinstimmung wird mit der Testdurchführung fortgefahren, andernfalls wird abgebrochen.

`<LOGFILE>`**:** Ist ein in "-Zeichen eingeschlossener Dateiname. Die Ausgabedatei (Protokolldatei o.ä.) des aufgerufenen Programmes wird durch Angabe ihres Namens in `<LOGFILE>` in das `\log\`-Verzeichnis des ATOS-Projekts verschoben.

Beschreibung: Startet ein externes Programm für testbegleitende Aufgaben (Dateivergleicher, Grafikviewer etc.), wartet auf dessen Terminierung und schreibt gegebenenfalls ein Logfile in das `\log\`-Verzeichnis des ATOS-Projektes.

Beispiel:

```
LAUNCH,BIN,"DATADIFF.EXE","C:\\TEST1.CRV TEST1.CRV.REF",FOREVER,
0,"DataDiff.log"
  # Der Dateivergleicher DataDiff wird aus dem \bin\-Verzeichnis
  # des ATOS-Projektes aufgerufen. Beim Vergleich der Dateien
  # C:\TEST1.CRV mit TEST1.CRV.REF wird die Protokolldatei
  # "DataDiff.log" angelegt und nach \log\ kopiert.
  # Die Durchführung des Skriptes wird fortgesetzt, wenn DataDiff
  # seine Arbeit mit dem Rückgabewert 0 beendet (kein Fehler).
```

2. Warten auf Zeit

Aufruf: `LAUNCH, <VERZEICHNIS>, <PROGRAMMNAME>, <PARAMETER>, TIME, <TERMINIERUNGSDAUER>, <RETURNCODE>, [,<LOGFILE>]`

Parameter:

`<TERMINIERUNGSDAUER>`**:** Gibt die Zeitlänge in Millisekunden an, die das aufgerufene Programm bis zu seiner Terminierung benötigt.

Alle anderen Parameter wie bei 1.

Beschreibung: Startet ein externes Programm für testbegleitende Aufgaben (Dateivergleicher, Grafikviewer etc.), wartet eine bestimmte Zeitlänge auf seine Terminierung und schreibt gegebenenfalls ein Logfile in das `\log\`-Verzeichnis des ATOS-Projektes.

Beispiel:

```
LAUNCH,BIN,"PROGRAMM.EXE","",TIME,2000,0
  # Startet PROGRAMM.EXE aus dem \bin\-Verzeichnis ohne
  # Parameter. Eventuell angelegte Ausgabedateien werden
  # nicht ins \log\-Verzeichnis kopiert. Nach spätestens
  # 2 Sekunden muss das Programm mit dem Rückkehrcode 0
  # terminieren.
```

3. Terminierung nicht abwarten

Aufruf: `LAUNCH, <VERZEICHNIS>, <PROGRAMMNAME>, <PARAMETER>, NOWAIT`

Parameter:

Parameter wie bei 1.

Beschreibung: Startet ein externes Programm für testbegleitende Aufgaben (Dateivergleicher, Grafikviewer etc.) und wartet nicht auf dessen Terminierung.

Beispiel:

```
LAUNCH,BIN,"i_view32.exe","TEST1.CRV.BMP",NOWAIT
  # Startet i_view32.exe aus dem \bin\-Verzeichnis mit
  # der Bitmap-Datei "TEST1.CRV.BMP" als Parameter.
  # Terminierung, Rückkehrcode und Ausgabedatei werden
  # nicht beachtet.
```

CLEANUP

Aufruf: `CLEANUP`

Parameter: keine

Beschreibung: Falls während des Testvorgangs ein Fehler auftritt, wird direkt zu diesem Kommando gesprungen, sofern es vorhanden ist. Die mit `START` aufgerufene Anwendung wird ggf. geschlossen. Alle folgenden Kommandos bis zum Ende der Datei dienen der Wiederherstellung des Zustandes vor Beginn der Ausführung des aktuellen Testskriptes. Temporär angelegte Dateien sollten hier gelöscht und originale Konfigurationsdateien wieder hergestellt werden, um das Testobjekt in seinen Ausgangszustand zu versetzen. Fehler beim Ausführen von Kommandos nach dem `CLEANUP`-Kommando werden ignoriert.

LOOP

Aufruf: `LOOP,<CYCLES>`

Parameter:

`<DAUER>`**:** Gibt die Anzahl der Schleifenzyklen an.

Beschreibung: Bildet den Anfang eines Schleifenblocks. Alle Kommandos zwischen `LOOP` und `ENDLOOP` werden bei jedem Zyklus wiederholt. Beliebig tiefe Verschachtelungen der Schleifenblöcke sind zulässig.

Beispiel:

```
LOOP,5
   # Fuehrt die hier eingefuegte Anweisungsfolge 5-mal aus
ENDLOOP
```

ENDLOOP

Aufruf: `ENDLOOP`

Parameter: keine

Beschreibung: Bildet das Ende eines `LOOP`-Schleifenblocks.

Beispiel: siehe Kommando `LOOP`

3.6 Entwurf von Konfigurationsdateien

Konfigurationsdateien sind Eingabedaten, die das Verhalten des XCTL-Systems wesentlich beeinflussen. Die Topographie arbeitet zum Beispiel mit anderen Motorbezeichnungen, als die Diffraktometrie. Grundsätzlich zerfallen die Konfigurationsdateien für die Hardware in diese beiden Kategorien. Für viele Testsequenzen können sogar die selben Dateien verwendet werden, wenn sie unabhängig von Hardware-Parametern sind oder gleiche Antriebs- und Detektor-Konfigurationen benutzen. Zur Unterscheidung der beiden Konfigurationskategorien wird der Dateiname um das Suffix `<n>TOPO_` bzw. `<n>DIFF_` erweitert, wobei `<n>` einen Index zur weiteren Differenzierung der Dateien einer Klasse kennzeichnet. Einige präparierte Konfigurationsdateien, wie zum Beispiel `OTOPO_HARDWARE.INI` und `ODIFF_HARDWARE.INI`, liegen breits im Verzeichnis `\env\` vor und finden in den spezifizierten Testfällen (siehe Kapitel 3.3.3) ihre Verwendung.

1. `DEVELOP.INI`

Die Konfigurationsdatei `DEVELOP.INI` ist für die Aufnahme nutzerspezifischer Informationen, wie Benutzername und Messprobenangaben, für die Festlegung von Fenstergrößen und Dialog-Defaultparametern, sowie für das Startverhalten des Steuerprogrammes verantwortlich. Wie sich herausstellen sollte, ist für den Entwurf beliebiger Testfälle zur Topographie und Diffraktometrie (Kapitel 3.3.3) eine einzige präparierte Konfiguration völlig ausreichend. Für diese Zwecke wurde die Datei `TEST_DEVELOP.INI` entworfen.

Die Einträge zur Positionierung der Fenster und zur Festlegung von Default-Parametern der Dialogboxen werden während der Verwendung einer Komponente angelegt bzw. überschrieben und sind daher für die Durchführung eines Regressionstestes nicht von Bedeutung. Zur Durchführung nicht automatisierbarer Testschritte und auch zur visuellen Überwachung der Sequenzen-Ausführung ist eine Positionierung und Größenangabe der Fenster dennoch sinnvoll. Ohne diese Angaben wäre jedes aufgerufene Fenster, einschließlich das Hauptfenster des Steuerprogrammes, in minimaler Größe. Die Datei `TEST_DEVELOP.INI` wurde dazu um die entsprechenden Einträge erweitert.

Für die Spezifizierung beliebiger Testfälle müssen aus dem XCTL-System heraus alle Programmfunktionalitäten aufrufbar sein. Dazu muss im Abschnitt `[Steuerprogramm]` der Eintrag `Environment=Expert` gesetzt werden. Außerdem sollen alle Fenster, Dialogboxen und Funktionen erst im

Zuge einer Testsequenz und nicht beim Programmstart geöffnet bzw. aufgerufen werden, weshalb zusätzlich die Einträge `Startup=Nothing` und `AutoCalibration=0` notwendig sind. Der Eintrag `CreateIniDefaults=1` ermöglicht das Zurückschreiben von Defaultparametern in die Konfigurationsdatei. Die eigentliche Durchführung der Testsequenzen wird davon nicht beeinflusst, wohl aber das Verhalten des Steuerprogrammes im Umgang mit der Ini-Datei, weshalb auch dieser Parameter in die präparierte Konfiguration aufgenommen wurde. Die Datei `TEST_DEVELOP.INI` entspricht somit im Gegensatz zur `<n>TOPO_HARDWARE.INI` bzw. `<n>DIFF_HARDWARE.INI` keiner gebräuchlichen Konfiguration eines Topographie- oder Diffraktometrie-Arbeitsplatzes. Der Spezifizierung von Testfällen werden auf diese Weise jedoch keine Grenzen gesetzt. Im Anhang C.1 ist das Ergebnis dieser Diskussion abgebildet.

2. HARDWARE.INI

Die Präparierung von Hardware-Konfigurationsdateien erfordert hingegen deutlich mehr Aufwand, weil ihre Einträge entscheidend für das Verhalten des XCTL-Systems sind. Zur Topographie und Diffraktometrie dienten uns dazu jeweils vier Konfigurationsdateien, welche an realen Arbeitsplätzen des Physik-Instituts eingesetzt werden. Diese Dateien liegen in ihrer Originalversion mit Parametern für die Detektoren und Antriebe, Benutzerinformationen sowie Einstellungen für Fenstergrößen und -positionen vor. Als Ergebnis einer Studienarbeit wurde das Prinzip von zentralistischen Konfigurationsdateien verändert. Das neue Inidateien-Konzept schreibt eine Trennung der nutzer- und hardwarerelevanten Parameter in zwei getrennte Dateien vor, weshalb zur Bildung von Hardware-Konfigurationen nur die Abschnitte (`[Device<n>]` und `[Motor<n>]`) untersucht wurden. Ziel war es, aus diesen vier Dateien repräsentative Konfigurationen für Topographie- und Diffraktometrie-Arbeitsplätze zu entwerfen. Die dabei entstandenen Dateien können dann als Vorlage zum Entwurf modifizierter Konfigurationsdateien für den Einsatz in neuen Testsequenzen dienen.

Im Kapitel 3.3.3 wird erkennbar, dass zur Spezifizierung von Testsequenzen die zwei 0-dimensionalen Testdetektoren `Counter` und `Simulant`, sowie der 1-dimensionale Testdetektore `PSD` völlig ausreichen. Die dazu notwendigen Abschnitte `[Device0]`, `[Device1]` und `[Device2]` wurden den Dateien `TOPO_HARDWARE.INI` und `DIFF_HARDWARE.INI` hinzugefügt. Wie im Anwendungsfall „Detektornutzung“ beschrieben, sind in der aktuellen XCTL-Version weitere Einträge nicht nur unnötig, sondern sogar unmöglich.

Zur Entwicklung der Abschnitte [Motor<n>] wurden alle Schlüssel/Wert-Paare aus den vier Konfigurationsdateien miteinander verglichen. Jeder Parameter wurde auf seine Bedeutung im XCTL-System und seine Relevanz innerhalb der Umgebungssimulation untersucht. Einige Schlüssel/Werte-Paare haben innerhalb der Umgebungssimulation keine Auswirkungen auf das Verhalten des Steuerprogrammes, einige Einträge entstammen aus früheren Programmversionen und werden ignoriert und einige Einträge sind im Laufe der Zeit sogar in falsche Abschnitte geraten.

Simulationsrelevante Parameter werden dagegen aus einer der vier Konfigurationsdateien übernommen. Dabei wird diejenige Konfigurationsdatei ausgewählt, welche die meisten Gemeinsamkeiten mit den anderen Dateien aufweist. Ergebnis dieses Verfahrens sind zwei realistische Arbeitsplatz-Konfigurationen TOPO_HARDWARE.INI und DIFF_HARDWARE.INI, die den Ausgangspunkt für präparierte Dateien zum Einsatz in beliebigen Testsequenzen bilden. Dabei ist die Abhängigkeit der Einträge untereinander nicht zu verachten. Die Einträge Unit und SpeedScale ergeben zum Beispiel nur zusammen eine sinnvolle Antriebs-Konfiguration und dürfen daher nicht aus verschiedenen Dateien übernommen werden.

Nach dem Zusammenstellen aller hardwarerelevanten Abschnitte wurden die Konfigurationsdateien außerdem um den Abschnitt [MOTORSIM] für die Motorensimulation, sowie um einen Abschnitt für einen Testmotor der alten Variante (Type=TMotor) erweitert. Ergebnis dieser Methode sind zwei „aufgeräumte“ Dateien, mit denen Testsequenzen für die zwei großen Teilbereiche Topographie und Diffraktometrie/Reflektometrie durchgeführt werden können. Für verschiedene zu testende Situationen sind nur noch wenige Modifikationen an den Dateien TOPO_HARDWARE.INI und DIFF_HARDWARE.INI notwendig.

Im Anhang C.2 ist die beschriebene Vorgehensweise dokumentiert. Die Tabellen stellen die gewonnenen Erkenntnisse bei der Untersuchung jedes Parameters auf seine Bedeutung im Steuerprogramm und Relevanz in Umgebungssimulation zusammen. Im folgenden sollen zwei Beispiele beschrieben werden, um die Ableitung spezieller Inidateien zu erläutern.

Beispiel 1: 1TOPO_HARDWARE.INI

Für die Testsequenzen zu den Anwendungsfällen „Detektoren“, „Automatische Justage“, „Topographie“, sowie „Motorsteuerung“ wurde die Datei 1TOPO_HARDWARE.INI abgeleitet und modifiziert. Ziel war es, eine bestimmte Grundstellung der Antriebe DF, Tilt und Kollimator ohne Zugriff auf die

„Manuelle Justage“ zu erreichen. Die Testsequenzen sollen relativ isoliert von anderen Anwendungsfällen durchgeführt werden und benötigen deshalb präparierte Konfigurationsdateien, welche das XCTL-System in einen bestimmten Ausgangszustand führen. Jede Modifikation sollte zusammen mit den bezweckten Absichten im Kopf der Inidatei als Kommentar dokumentiert werden. Alle Anwendungsfälle, in denen diese Datei eingesetzt wird, sollten dort ebenfalls notiert werden. Schließlich dient ein Datum als letzter Eintrag des Kopfes zur Überwachung der Aktualität.

An den Einträgen für die Detektoren waren keine Modifikationen nötig. Der `Name` „Beugung Fein“ wurde auf „DF“ geändert, um den Antrieb mit der für die Topographie typischen Bezeichnung anzusprechen. Für einen Test der Antriebs-Initialisierung, wurde der Wert `InitialAngle` des Antriebs „DF“ auf `60.0` gesetzt. Außerdem wurden die Werte für `DeltaPosition` bei den Antrieben „DF“, „Tilt“ und „Kollimator“ so modifiziert, dass die Intensität des 0-dimensionalen Testdetektors im Zusammenspiel mit der virtuellen Testprobe (`TESTDEV.DAT`) auf einem Peak steht. Da der Abschnitt zum Antrieb `Kollimator_neu` für die Durchführung der Testsequenzen keine Rolle spielt und eventuell zu Verwirrungen mit dem eingesetzten Antrieb `Kollimator` führen könnte, wurde er aus dieser Konfigurationsdatei herausgenommen.

Beispiel 2: 0DIFF_HARDWARE.INI

Die Datei `0DIFF_HARDWARE.INI` wurde für den Einsatz in Testsequenzen zu den Anwendungsfällen „LineScan“ und „AreaScan“ innerhalb des UseCases „Diffraktometrie/Reflektometrie“ entworfen. Hierbei ging es weniger um die Ausgangsstellung der Motoren, als vielmehr um die Modifikation der Parameter für Geschwindigkeit und Beschleunigung. Die übernommenen Werte von `(Max)Velocity` und `Acceleration` führen durch die Realitätsnähe der Motorensimulation zu sehr langsamen Bewegungen. Eine Testsequenz, die zum Beispiel einen bestimmten Winkelbereich durchlaufen soll, um eine Messkurve aufzunehmen, würde mehrere Minuten in Anspruch nehmen. Der Regressionstest sollte jedoch möglichst zügig durchgeführt werden können. Das Verhalten der beiden Motoren Theta und Omega wird so modifiziert, dass sie tausendfach schneller laufen und schneller beschleunigen, als unter realen Bedingungen. Man sollte jedoch dabei beachten, dass Modifikationen an der Konfiguration der Antriebshardware zu verheerenden Konsequenzen führen können und daher nur im Rahmen der Motorsimulation angewendet werden sollten!

Um realitätsnahe Kurven für Messungen über die Antriebe „Omega“ und „Theta“ zu erhalten, wird wieder auf den 0-dimensionalen Testdetektor zugegriffen, der seine Messwerte aus der virtuellen Testprobe `TESTDEV.DAT` bezieht. „Omega“ und „DF“ werden programmintern gleichwertig behandelt, einen Kollimator gibt es dagegen an den Diffraktometrie-Arbeitsplätzen normalerweise nicht. Trotzdem wird ein entsprechender Eintrag der Konfigurationsdatei `ODIFF_HARDWARE.INI` hinzugefügt. Mit der richtigen Positionierung des Kollimators können aufschlussreiche und realitätsnahe Messkurven über einen bestimmten Omegabereich simuliert werden.

3.7 Entwurf von Referenzdateien

3.7.1 Ausgabedatenvergleich mit *DataDiff*

Diese Testkomponente vergleicht Ausgabedateien des XCTL-Systems unterschiedlichen Formates miteinander, toleriert dabei Unschärfen und erstellt ein Protokoll über den Erfolg seiner Arbeit. In Tabelle 3.4 sind neun verschiedene Datenformate des XCTL-Systems (Version vom 20.01.2002) aufgelistet.

Dateiendung	**Inhalt**	**Datenstruktur**

`.ini`	Konfigurationsdatei mit Schlüssel- und Werteinträgen
	`develop.ini`
		`[Abschnitt1]`
		`Schlüssel1=Wert1`
		`Schlüssel2=Wert2`
		`[Abschnitt2]`
		`Schlüssel3=Wert3`
		`Schlüssel4=Wert4`

`.log`	Logdatei der „Automatischen Justage“ `Justage.log`	Verbaler Text beschreibt Verlauf der Justage Schlüsselworte: `Optimierung`, `Nachregeln`, `Intensität` ...
	Logdatei der „Makroausführung“ `MACRO.LOG`	Verbaler Text beschreibt Verlauf der Makroausführung Schlüsselworte: `ChooseAxis`, `MoveToPoint`, `D:5.005 I:22555.00` ...

	Logdatei der ”Detektornutzung“
	`device0.log`
		Startzeitpunkt der Protokollierung
		`590.000`
		`586.000`
		`583.000` ...

`.bk`	Vergleichs-Scan bei „Diffraktometrie“	Header mit Informationen über Spalteninhalte
		`[Data]`
		`6.500 1271.000 0.000`
		`7.000 1268.000 0.000` ...

`.crv`	Scandaten der „Diffraktometrie“ („LineScan“)	Header mit Informationen über Benutzer und Messprobe
		`[Data]`
		`-5.00000 865.0000 0.0000`
		`-3.00000 853.0000 4.0000` ...

`.psd`	Scandaten der „Diffraktometrie“ („AreaScan“)	Header mit Informationen über Benutzer und Messprobe `[Data]` `8.64 9.60 5.76 17.3 7.68` ...
`.rep`	Reportdatei über den Scanvorgang der „Diffraktometrie“ („AreaScan“)	Header mit Informationen über Spalteninhalte `0 -1.0000 0.000 1148 2044.000` ... `1 -0.8000 0.400 1141 2044.400` ...
`.dtn`	Datenerhebungsdatei	Header mit Informationen über Benutzer und Messprobe `[Data]` `0.21511397 4.21430016 1.62898497` ... `0.21577907 4.21430016 1.62686784` ...

Tabelle 3.4: Ein- und Ausgabedateien des XCTL-Systems

Wie aus Tabelle 3.4 ersichtlich, unterscheiden sich Struktur und Inhalt der Ausgabedateien sehr stark voneinander. Möglicherweise werden im Laufe der Projektarbeit sogar einige Dateien in ihrer Struktur verändert. Die Reportdateien des „AreaScan's“ wurden zum Beispiel kürzlich um die Spalten `MaxIntensity`, `PeakPosition` sowie `MeasurementTime` erweitert.

Ein Vergleich mit einer Solldatei aus früheren Programmversionen würde dann zu Fehlern führen. Werden Änderungen an Inhalten oder Strukturen durch eine Aktualisierung des XCTL-Systems hervorgerufen, ist die dafür verantwortliche Entwicklergruppe verpflichtet, aktuelle Solldateien für entsprechende Testsequenzen zu erstellen und dem Verzeichnis `\ref\` hinzuzufügen.

DataDiff bekommt über die Konsole zwei Parameter überreicht. Der erste Parameter benennt die zu prüfende „Istdatei“, die während des Durchlaufs einer Testsequenz entstand. Der zweite Parameter bestimmt die zum Vergleich herangezogene „Solldatei“. Mit dem optionalen Schalter `/s` lässt sich die Protokollausgabe auf dem Bildschirm unterdrücken, um den Vergleichsvorgang zu beschleunigen. Als eigenständige Konsolenanwendung ist *DataDiff* völlig unabhängig von der Testsuite. *ATOS* ruft im Laufe seiner Arbeit bei Bedarf dieses Werkzeug als Prozess auf und übergibt ihm die notwendigen Parameter. Es ist leicht möglich, den Dateivergleich mit anderen Werkzeugen durch ein entsprechendes `LAUNCH`-Kommando (siehe Skriptkommandos, Kapitel 3.5.3) durchzuführen.

3.7.2 Vergleichskommandos

Referenzdateien und Istdateien können nicht 1:1 verglichen werden. Schon das abgespeicherte Datum einer Messung führt beim Vergleich mit einer Solldatei aus früheren Testläufen zu Widersprüchen. Intensitätswerte der Testdetektoren schwanken um einen zufälligen Anteil und führen zu leichten Abweichungen in den ausgeschriebenen Daten bei wiederholten Messungen, trotz gleicher Bedingungen. Sogenannte Metainformationen müssen zusätzlich dem Vergleichsprogramm Hinweise geben, wie mit den vorliegenden Daten umzugehen ist. Einige Zeichenketten oder sogar ganze Zeilen sind bei der Gegenüberstellung von Ist- und Solldatei auszulassen. Andere Zeichenketten müssen als numerische Werte interpretiert und mit Toleranzen verglichen werden.

Zur einfachen Wartung werden Metadaten (Kommandos) und Vergleichsdaten in einer einzigen Datei gehalten. Referenzdateien, die keine Anweisungen für *DataDiff* enthalten, werden wie gehabt 1:1 verglichen. Prinzipiell

werden in Solldateien vorgefundene und zusammenhängende alphanumerische Zeichenketten in den Istdateien ab der aktuellen Lese-Position aufgesucht. Diese Lese-Position kann mit speziellen Anweisungen, wie in Tabelle 3.5 dargestellt, beeinflusst werden. Zeichenketten, die in den Solldateien nicht vorkommen, werden auch nicht in den Istdateien gesucht.

Die folgenden Kommandos werden beim Vergleich von Textdateien berücksichtigt.

Anweisung	**Auswirkung**
$nextline$	Überspringt den Rest einer Zeile. Setzt Such-Position auf nächste Zeile.
$somewhere$	Nachfolgende Zeichenkette darf an beliebiger Stelle in der Istdatei vorkommen. Such-Position wird danach wieder auf Ausgangsposition gesetzt.
$back$	Nachfolgende Zeichenkette darf an beliebiger Stelle in der Istdatei vorkommen. Such-Position wird hinter die gefundene Zeichenkette gesetzt.
$col <n1> <n2> ...$	Beginn eines numerischen Vergleichs von Datenwerten. Die folgenden Daten werden zyklisch unter Berücksichtigung von Toleranzen (<n1>, <n2> etc.) bis zum Ende der Datei oder bis $endcol$ verglichen.
$endcol$	Beendigung des zyklischen und numerischen Vergleichs.
eq	Nachfolgende Zeichenkette wird numerisch verglichen. Istwert muss gleich Sollwert sein.
leq	Nachfolgende Zeichenkette wird numerisch verglichen. Istwert muss kleiner gleich Sollwert sein.
geq	Nachfolgende Zeichenkette wird numerisch verglichen. Istwert muss größer gleich Sollwert sein.
lss	Nachfolgende Zeichenkette wird numerisch verglichen. Istwert muss kleiner Sollwert sein.
grt	Nachfolgende Zeichenkette wird numerisch verglichen. Istwert muss größer Sollwert sein.
$tol <n>$	Nachfolgende Zeichenkette wird numerisch verglichen. Istwert muss innerhalb Sollwert ±<n> liegen.
//	Bis zum Ende der Zeile werden alle Zeichenketten als Kommentar interpretiert und beim Vergleich ignoriert.

Tabelle 3.5: Vergleichsanweisungen - *DataDiff*

Das Beispiel in Abbildung 3.6 zeigt die Referenzdatei, die beim Ausführen des Testskriptes `Test_ARS.2.HTS` zum Vergleich einer Ausgabedatei herangezogen wird. Beim zyklischen Vergleich nach `$col 0 6 0$` müssen alle Werte aus der ersten und letzten Spalte exakt mit den Erwartungswerten übereinstimmen. Der Intensitätswert in der zweiten Spalte darf hingegen um ±6 vom Sollwert abweichen. Wo Abweichungen zulässig sein sollen, sind häufig

```
// Referenzdatei für den Testfall ARS.2
// Anwendungsfall Diffraktometrie/Reflektometrie -> AreaScan
// 07.02.2002


[Header]
Zerlegung=SLD
Point_Number= $eq$ 5
FileType=Standard
ArgumentMin= $eq$ -0.5000
ArgumentWidth= $eq$ 0.20000
ArgumentMax= $eq$ 0.3000
Scanaxis=Theta
TimePerScan= $eq$ 1.00

[Data]
$col 0 6 0$
-0.50000  774.0000  -1.0000
-0.30000  774.0000  -1.0000
-0.10000  774.0000  -1.0000
0.10000   774.0000  -1.0000
0.30000   774.0000  -1.0000
```

Abbildung 3.6: Solldatei `TEST0000.CRV.REF`

Mittelwerte in den Referenzdateien zu bilden. Für die Testdetektoren muss dazu der Berechnungsalgorithmus zur Intensitätsbestimmung ohne zufälligen Anteil aufgerufen werden. Bei Benutzung des 0-dimensionalen Testdetektors, der seine Intensitäten aus der Datei `TESTDEV.DAT` einliest, genügt ein Blick auf die jeweilige Antriebsposition.

Ein kleines selbstgeschriebenes Programm berechnet gemittelte Intensitätswerte für beliebige Parameterkonfigurationen (Theta, Omega, Messzeit etc.) des 0-dimensionalen Testdetektors von Hr. Damerow und des 1-dimensionalen Testdetektors. Dazu werden die gleichen Algorithmen der Hardwaresimulatoren aus dem XCTL-System ohne Einberechnung eines Zufallwertes verwendet. Das Programm ist im Verzeichnis `.\src\Simulant\Release\` nach dem Auscheken von `ATOS` aus dem CVS-Repository vorzufinden (siehe Abschnitt 3.2.1). Der Aufruf der MSDOS-Konsolenanwendung `Simulant.exe` gibt Aufschluss über die Verwendung ihrer möglichen Parameter.

3.8 Unterstützung bei der Skriptentwicklung

Wie im Kapitel 3.3 beschrieben, wurden alle Testfälle aus den Verhaltensspezifikationen der Anwendungsfälle gewonnen. Aus den Tabellen im Web-Repository entstehen Testskripte durch Übersetzung der verbal beschriebenen Testschritte in Kommandos unserer entwickelten Skriptsprache HTS. Dafür muss der Entwickler von Testsequenzen sowohl die Kommando-Syntax als auch alle verfügbaren Ressourcen (Oberflächen-Elemente) des Testobjektes kennen. Wünschenswert wäre außerdem ein Verfahren zur systematischen und automatisierten Gewinnung von Testsequenzen.

Zur Behandlung der letzteren Problematik wird die Klassifikationsbaum-Methode eingesetzt. Die manuelle Konstruktion von Testsequenzen in der Skriptsprache wird durch eine dialoggesteuerte Komponente aus der Testsuite *ATOS* unterstützt.

3.8.1 Die Klassifikationsbaum-Methode

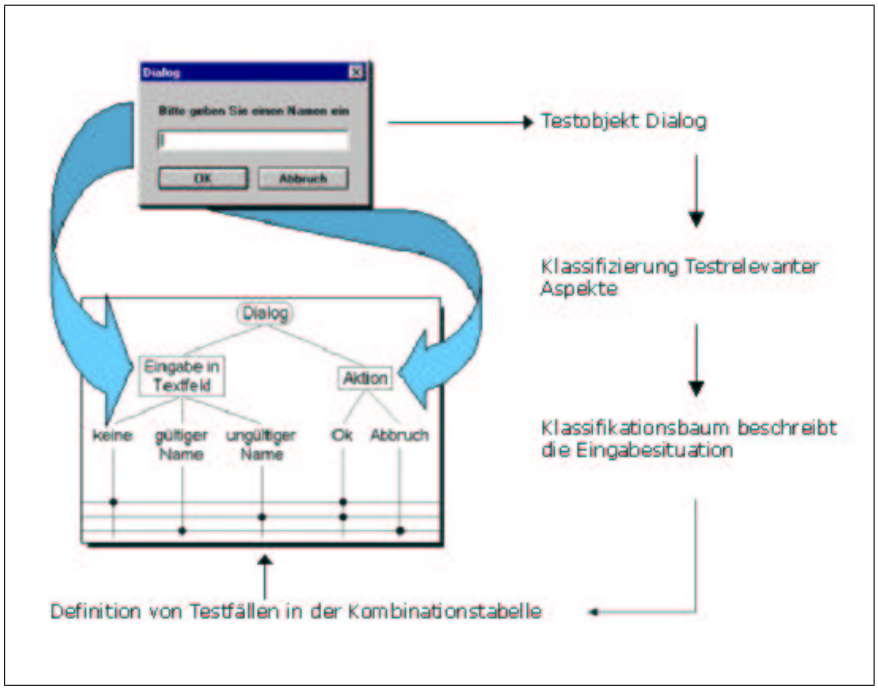


Abbildung 3.7: CTE Vorgehensmodell (Quelle: www.razorcat.de)

Einen wesentlichen Teil dieser Arbeit bildet die Untersuchung von Möglichkeiten zur systematischen und automatisierten Gewinnung von Testfällen. Durch die Diplomarbeit von Stefan Lützkendorf [10] inspiriert, entschlossen wir uns den *Classification Tree Editor* von *DaimlerChrysler AG* und *Razorcat Development GmbH* zur Lösung dieser Aufgabe einzusetzen. Mittels Zerlegung der Eingabedaten in Äquivalenzklassen können Testfälle und Testsequenzen systematisch entworfen werden.

Alle Eingaben, die einen Aspekt gemeinsam haben, bilden eine Klassifikation. Im Anwendungsfall „Manuelle Justage“ sind zum Beispiel die Klassen „Direktbetrieb“, „Schrittbetrieb“ und „Fahrbetrieb“ der Klassifikation „Art der Bewegung“ zuzuordnen. Im Laufe der Projektarbeit sind schon für einige Anwendungsfälle solche Klassifikationsbaum-Diagramme erfolgreich entworfen worden. Es liegt also Nahe, dieses Werkzeug auch im Rahmen der Diplomarbeit zur Gewinnung von Testsequenzen für unser Testsystem einzusetzen.

Der Klassifikationsbaum besteht aus einem Wurzelelement, aus dem sich Klassifikationen und Klassen im Wechsel verzweigen. Alle Klassen einer Klassifikation sind disjunkt, d.h. in einem Testfall kann höchstens eine der Klassen ausgewählt werden. Die Auswahl der Klassen erfolgt in einer Kombinationstabelle unterhalb des Klassifikationsbaumes (siehe Abbildung 3.8). Der Entwurf eines CTE-Diagrammes erfolgt demnach in zwei Schritten. Zunächst wird ein Baum mit allen Klassifikations- und Klassenelementen konstruiert. Erst wenn die Testdatenmenge des untersuchten Anwendungsbereichs in ausreichend viele Äquivalenzklassen zerlegt wurde, kann mit der Definition von Testfällen in der Kombinationstabelle begonnen werden. Eine minimale Überdeckung, also die Auswahl jeder Klasse in wenigstens einem Testfall, sollte dabei erreicht werden.

Die CTE-Benutzeroberfläche ist in drei Fensterbereiche aufgeteilt.

Die **Klassifikationsbaum-Zeichenfläche** dient zur Erstellung eines Klassifikationsbaumes. Klassifikations- bzw. Klassenelemente können schrittweise aus Vaterelementen angelegt werden oder nachträglich mit Vaterelementen verknüpft werden. Mit „Refinements“ ist es sogar möglich, das Diagramm in weitere Unterdiagramme zu strukturieren, um die Übersichtlichkeit zu bewahren. Sogenannte „Kompositionen“ weichen das strenge Wechselprinzip von Klassifikationen und Klassen in den Ebenen auf. Kompositionen können in beliebig viele weitere Kompositionen verzweigen. Alle verknüpften Elemente zusammen bilden den Klassifikationsbaum.

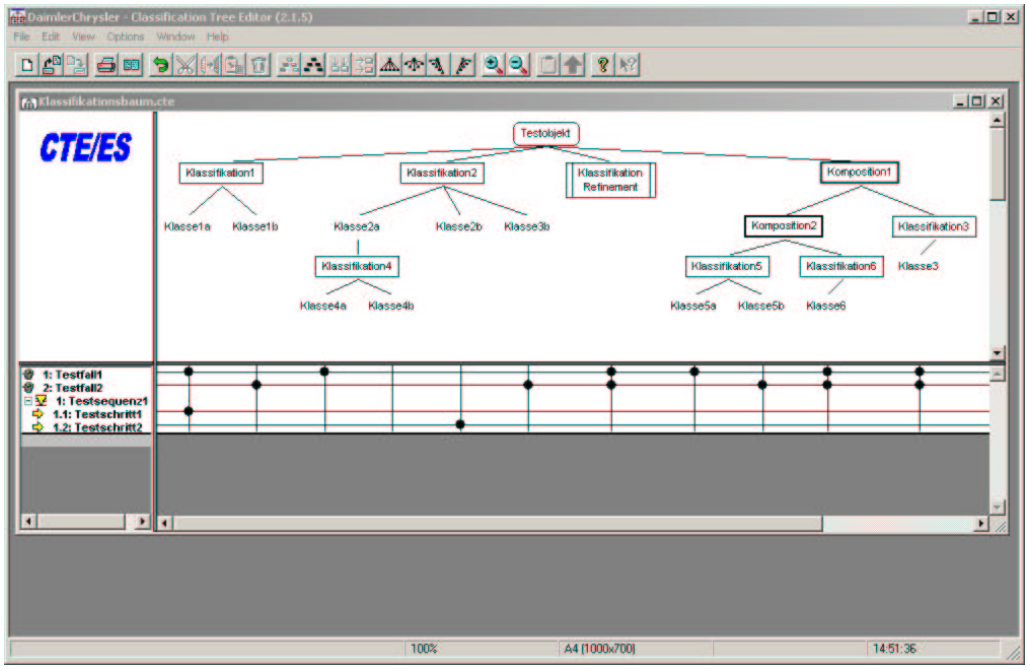


Abbildung 3.8: Classification Tree Editor

Das **Testfall/-sequenz-Listenfeld** dient zur Eingabe und Verwaltung von Testfällen, Testsequenzen und Testschritten. Hierbei ist die Unterscheidung zwischen Testfall und Testsequenz genau zu beachten. Mit der Auswahl von Klassen in Testfällen wird eine bestimmte Situation des Testobjektes wiedergegeben, die mit dem Testfall überprüft werden soll. Eine Testsequenz kann als eine Menge von Testfällen verstanden werden, die nacheinander ausgeführt werden. Das Testobjekt wird dann von einer Situation in die nächste überführt. Ob Testfälle oder Testsequenzen in einem Diagramm eingesetzt werden, hängt mitunter stark von der Konstruktion des Klassifikationsbaumes ab. Im Abschnitt 3.8.1 wird darauf genauer eingegangen. Jeder Testfall bzw. Testschritt wird im Testfall/-sequenz-Listenfeld automatisch durchnummeriert und erzeugt in der dazugehörigen Kombinationstabelle eine Tabellenzeile.

In der **Kombinationstabelle** erfolgt durch das Setzen von Markierungen das Spezifizieren der Testfälle und Testschritte. Sobald sich der Mauszeiger in der Kombinationstabelle befindet, werden Verbindungslinien zum Klassifikationsbaum gezeichnet. Um Klassen eines Testfalles bzw. Testschrittes auszuwählen, werden die Schnitpunkte zwischen den Tabellenzeilen und Tabellenspalten mit der linken Maustaste markiert.

Attributierte Klassifikationsbäume

Die grundsätzliche Idee, Testschritte in unserer Skriptsprache in die Attribute der Baumelemente einzufügen, um sie beim Generieren von Testfällen durch einen Diagramm-Parser in geordneter Reihenfolge in Testskripte auszuschreiben, haben wir aus der Diplomarbeit von Herrn Lützkendorf aufgegriffen.

Wir entschieden uns für die Eigenentwicklung eines Diagramm-Parsers, bei dem Benutzerfreundlichkeit an oberster Stelle steht. Der Parser soll als dialoggesteuerte Komponente in unser Testsystem eingearbeitet werden, um einen Import von Testfällen aus attributierten CTE-Diagrammen zu ermöglichen. Dafür ist das kommandozeilenorientierte Perlprogramm von Herrn Lützkendorf leider nicht geeignet.

Weiterhin sollte sich die Komponente zum Parsen von CTE-Diagrammen in die objektorientierte Klassenstruktur des Gesamtprojektes einordnen, welches ausschließlich mit Microsoft's Visual C++ implementiert wurde. Die CTE-Diagramme werden in einer überschaubaren Baumstruktur als Textdateien abgespeichert und eigenen sich deshalb hervorragend zur Weiterverarbeitung mit bereits entwickelten Basiskomponenten bzw. -klassen unserer Testsuite.

Nicht alle Konzepte des Diagramm-Parsers von Herrn Lützkendorf wurden in unserem Parser umgesetzt. Einige neue Ansätze mussten dagegen zur Lösung der Aufgabe hinzugenommen werden. Im folgenden werden Gemeinsamkeiten und Unterschiede beider Implementierungen gegenübergestellt.

Gemeinsamkeiten

Skriptkommandos und Anweisungen zur Erstellung von INI-Dateien stehen in den Attributen der Baum-Elemente und werden durchnummeriert. (`step|1` ... `step|2` ... `ini|...`) Dabei erfolgt die Skriptgewinnung durch ein Top-Down-Verfahren: In jedem Testschritt bzw. Testfall (Zeile in der Kombinationstabelle) werden alle Pfade von der Markierung bis zur Wurzel verfolgt, um von dort aus das Skript beim Herunterlaufen und Auswerten der Attribute zu erstellen. Alle Elemente, die dabei durchlaufen werden, sind an der Konstruktion eines Testfalles bzw. Testschrittes beteiligt und werden im Weiteren als „konstruierende Elemente“ bezeichnet.

Die Nummerierung der Schritte sorgt für die Ordnung der ausgeschriebenen Testschritte in das erzeugte Skript. Anweisungen zur Ausgabe von Schlüssel/Wert-Paaren in INI-Files unterliegen keinerlei Ordnung, können

aber sehr wohl von Tochterelementen überschrieben werden. Testschritte können innerhalb eines Zweiges im Baum von den Tochterelementen von Testschritten mit gleicher Nummer überschrieben werden. Ebenso können Variablen definiert und von Tochterelemente überschrieben werden. Dieses Verhalten ist als Attribut-Vererbung in der Arbeit von Herrn Lützkendorf beschrieben worden.

Unterschiede

- Es gibt nur einen Variablentyp mit globaler Sichtbarkeit. Wie Herr Lützkendorf selbst feststellte, sind Variablen mit lokaler Sichtbarkeit (innerhalb eines Zweiges) nicht praxisrelevant und werden deshalb nicht unterstützt.

- Rechenoperationen auf Testdaten können mit den Möglichkeiten der Skriptsprachen durchgeführt werden. Einen Anwendungsfall, für den Rechenoperationen mit Attributen auf Ebene der CTE-Diagramme notwendig wären, konnten wir nicht konstruieren, weshalb auf die Fähigkeit des Diagramm-Parsers zur Auswertung von mathematischen Ausdrücken verzichtet wird.

- Das Konzept zur Formulierung von Zusicherungen für bestimmte Bedingungen bei der Testsequenz-Generierung (`assert`) ist überflüssig, wenn die Klassifikationsbäume widerspruchsfrei konstruiert werden. D.h. widersprüchliche Testfälle, die durch Kombinationen von sich ausschließenden Klassen entstehen, dürfen erst gar nicht gebildet werden können. Die Konstruktion von Klassifikationsbäumen muss demnach sorgfältig vorgenommen werden, da unser Diagramm-Parser das `assert`-Konzept nicht unterstützt.

- Unser Skriptgenerator bzw. Diagramm-Parser kennt generell keine vordefinierten Variablen, wie `%test|step_nr`, `%test|pred_step`, `%Skriptgenerator` oder `%test_id`. In der Diplomarbeit von Herrn Lützkendorf [10] werden die Bedeutungen dieser Variablen für seine Aufgabe deutlich.

- Die zwei Attribute `first` und `last` erweitern die Möglichkeiten von `step`. So können auch Positionierungen von Testschritten vor oder hinter ein schon zusammengestelltes Skript erfolgen. Genaueres dazu folgt im nächsten Abschnitt.

Testfallentwurf

Der Erfolg bei der systematischen Gewinnung von Testfällen beginnt beim Entwurf eines sinnvollen Klassifikationsbaumes. Sehr konzentriert müssen weiterhin die Attribute zur Definition von Testschritten, Variablen und INI-Parametern in den Elementen vorgenommen werden. Ein kleiner Fehler bei der Attributüberladung oder bei der Spezifizierung der Skriptkommandos kann zu unerwarteten Testskripten führen. Liegt einmal ein widerspruchsfreier Klassifikationsbaum vor, können beliebige Testfälle bzw. Testsequenzen in der Kombinationstabelle zusammen gestellt und in äquivalente Skripte zur Ausführung in *ATOS* übersetzt werden.

Die folgenden Attribute können in beliebigen Baum-Elementen verwendet werden:

1. `step|<n>`
   Mit dem Schlüsselwort `step` werden Skriptkommandos gekennzeichnet. Hinter '`|`' darf ein beliebiger positiver ganzzahliger Wert stehen. `<n>` gibt die Position des Kommandos innerhalb des erzeugten Skriptes an. Achtung, in Testsequenzen gilt die Nummerierung für jeden einzelnen Testschritt !

2. `first|<n>`
   In CTE-Diagrammen mit Testsequenzen würde für jeden Testschritt, der ein konstruierendes Element mit `step`-Attributen auf seinem Pfad von der Markierung bis zur Wurzel hat, ein entsprechendes Skriptkommando ausgeschrieben werden. Um das zu verhindern oder um in beliebigen Elementen Skriptkommandos vor alle anderen zu stellen, kann das Schlüsselwort `first` verwendet werden. Es verhält sich in seiner Syntax ansonsten wie `step`.

3. `last|<n>`
   Dieses Schlüsselwort ist das Pendant zu `first` und ermöglicht das einmalige Setzen von Kommandos an das Ende des Skriptes.

4. `ini|<INI-File>|<Abschnitt>|<Schlüssel>`
   Mit dem Schlüsselwort `ini` ist es möglich, INI-Dateien anzulegen und mit bestimmten Schlüssel/Wert-Paaren zu belegen. Wie schon erwähnt zählen die INI-Parameter des XCTL-Systems zu den Eingabedaten beim Testen und haben einen erheblichen Einfluss auf das Verhalten des Programmes. `<INI-File>` ist der Dateiname der Konfigurationsdatei. `<Abschnitt>` ist der `[Abschnitt]`, unter dem die Schlüssel/Wert-Paare in der Datei eingetragen werden sollen.

5. `%<Variable>`
   Mit dem `%`-Zeichen werden Variablen definiert. Der Bezeichner `<Variable>` darf aus beliebigen, alphanumerischen Zeichen bestehen.

Bei der Definition von Attributen sind folgende Regeln einzuhalten:

1. Elemente des Klassifikationsbaumes erben Testschritte, Variablendefinitionen und INI-Parameter aus den Attributen der Vaterelemente. Testschritte (`step`'s) mit gleicher Nummer dürfen nur einmal in einem Pfad durch Tochterelemente überschrieben werden. Es ist nicht erlaubt, einen Testschritt mit gleicher Nummer in Elementen auf verschiedenen Pfaden bis zum Wurzelelement zu definieren, die zusammen an der Konstruktion eines Testschrittes bzw. Testfalles beteiligt sind (in Kombinationstabellenzeile ausgewählt). Für INI-Parameter gilt das Gleiche. Bei Variablen gilt die Auflösungsregel (Punkte 2 und 3).

2. Bei der Auflösung von Variablen in Skriptkommandos oder INI-Parametern wird folgendermaßen vorgegangen: Zunächst wird in den Attributen aller Elemente vom konstruierenden bis zum Wurzelelement nach einer Definition gesucht. Weiterhin wird mit allen weiteren Klassen-Elementen, die den Testfall bzw. Testschritt konstruieren, genauso verfahren. Kann eine Variable nicht aufgelöst werden, kommt es beim Parsen zu einer Fehlermeldung.

3. Variablen dürfen auch in Testsequenzen, -schritten und -fällen (im Testfall/-sequenz-Listenfeld) definiert werden. Dabei überschreiben gleichnamige Variablenbezeichner die Attribute der konstruierenden Baum-Elemente. Variablen der Testschritte überschreiben gleichnamige Variablen der umfassenden Testsequenz.

4. Die Attributdefinitionen müssen schleifenfrei sein. Bei der Verwendung von Variablen ist darauf zu achten, dass die Auflösung in einem Testschritt bzw. Testfall nicht zu einer Endlosschleife führen kann.

Wie ein Klassifikationsbaum unter Berücksichtigung dieser Regeln attributiert werden kann, soll im nächsten Abschnitt anhand eines Beispiels für den Test der Dialogbox „AreaScan-Setup“ erläutert werden. Wichtig ist zunächst die Bemerkung, dass sich nicht jeder Klassifikationsbaum zur Attributierung eignet. Oft ist es schwer, für ein Testobjekt ein geeignetes Diagramm zu finden, welches gleichzeitig den Ansprüchen eines gründlichen Tests der Funktionalitäten genügt und sich für eine automatisierte Skriptgenerierung präparieren lässt.

Bei der Konstruktion eines CTE-Diagrammes sollte daher zuvor überlegt werden, ob man das Design so gestaltet, dass es sich zur Spezifikation von Testsequenzen eignet oder ob man die Strukturierung der Klassifikationen und Klassen eher zur Spezifikation von Testfällen - also von Testsituationen - vornimmt.

Testfälle, wie sie im Kapitel 3.3 entworfen wurden, ließen sich nur mit Testsequenzen in CTE-Diagrammen nachbilden, da hierbei die automatische Generierung eines solch komplexen Testskriptes durch Markierungen von Klassen-Elementen in einer einzigen Zeile in der Kombinationstabelle nahezu unmöglich ist.

Zur Veranschaulichung der Problematik sollen die zwei folgenden Beispiele dienen, die im Rahmen des Projektes „Software-Sanierung“ entstanden sind. Die Abbildungen 3.9 und 3.10 zeigen ein CTE-Diagramm für den Anwendungsfall „Manuelle Justage“. Die Klassen unterhalb der Klassifikationen beschreiben bestimmte Zustände, in denen sich die zu testende Komponente befinden kann. Die Kombinationen in der Tabelle beschreiben dementsprechend Testzustände, also Testfälle, die untersucht werden sollen. Diesen Klassifikationsbaum mit Skriptkommandos so zu attributieren, dass die Testfälle realisiert werden, ist eine sehr schwierige Aufgabe.

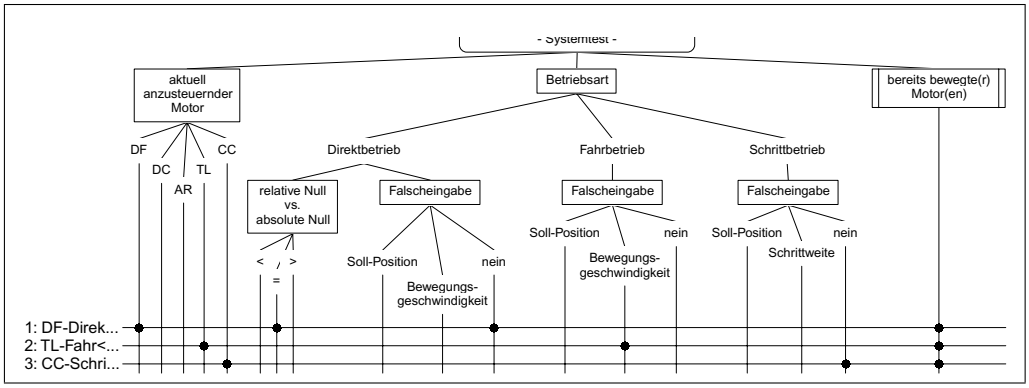


Abbildung 3.9: CTE-Diagramm zur „Manuellen Justage“

Die Baumstruktur lässt widersprüchliche Testfälle zu. So kann als aktuell anzusteuernder Motor zum Beispiel 'AR' ausgewählt werden. Im Unterdiagramm zur Festlegung bereits bewegter Antriebe (siehe Abbildung 3.10) kann bei 'AR' der „Schrittbetrieb“ festgelegt werden. Paradoxer Weise läßt sich für den selben Antrieb seine gegenwärtige Betriebsart auf „Direktbetrieb“

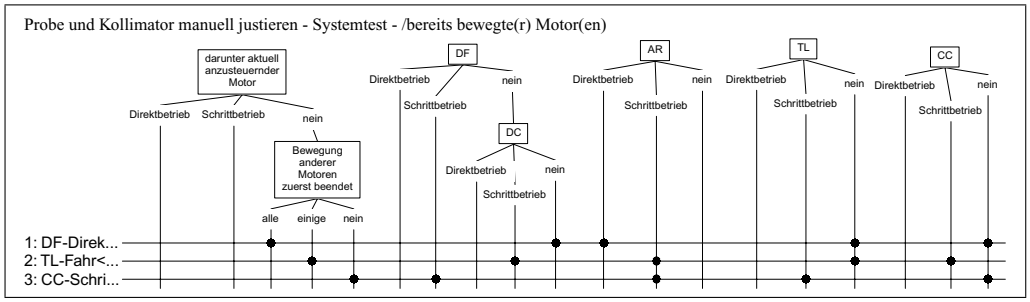


Abbildung 3.10: CTE-Diagramm zur „Manuellen Justage“ (bewegte Motoren)

in der Klassifikation „darunter aktuell anzusteuernder Motor“ setzen. Widersprüchliche Testfälle führen in attributierten Diagrammen zu fehlerhaften Testsequenzen. Sie lassen sich vermeiden, indem das Diagramm widerspruchsfrei konstruiert wird.

Schwierig stellt sich zudem die Realisierung der Klassifikation „Bewegung anderer Motoren zuerst beendet“ dar. Die Klasse „einige“ lässt sich nicht in konkrete Skriptkommandos übersetzen.

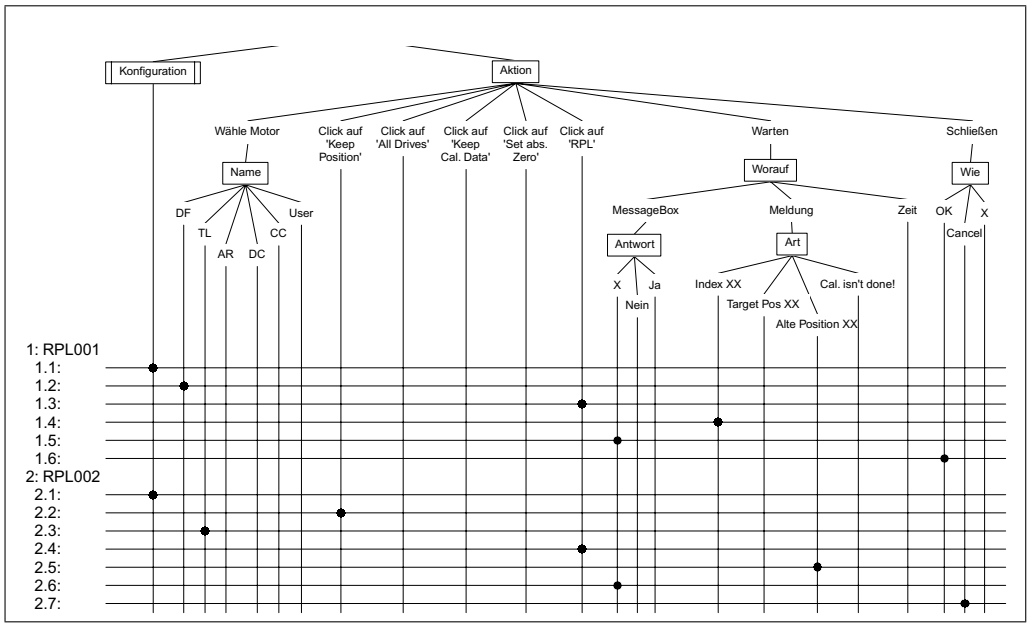


Abbildung 3.11: CTE-Diagramm zum „Referenzpunktlauf“

Eine völlig andere Situation beschreibt das CTE-Diagramm in Abbildung 3.11. Die Klassifikationen strukturieren das Diagramm nicht nach bestimmten Zustände, sondern nach möglichen Aktionen, welche mit der Dialogbox zum „Referenzpunktlauf“ durchgeführt werden können. Dementsprechend sind in der Kombinationstabelle Markierungen für Testschritte einer Testsequenz zu finden.

Die Attributierung eines solchen Diagrammes gestaltet sich wesentlich einfacher, da es aktionsorientiert und nicht zustandsorientiert ist. In den jeweiligen Attributen der Elemente sind lediglich die äquivalenten Skriptkommandos zur Ausführung der gewünschten Aktion zu setzen. Schwierigkeiten mit der Ordnung der Skriptkommandos für jeden einzelnen Testschritt treten nur selten auf, weil in jeder Kombinationszeile (Testschritt) meistens nur eine einzige Element-Klasse des Klassifikationsbaumes ausgewählt wird. Die tatsächliche Reihenfolge der erzeugten Skriptkommandos erfolgt durch die Anordnung der Testschritte im Testfall/-sequenz-Listenfeld.

Beispiele für zustandsorientierte Diagramme

Geeignete CTE-Diagramme für eine Komponente zu konstruieren, ist schwierig, vorhandene CTE-Diagramme zu attributieren, ist noch schwieriger. Um diese Möglichkeit zu untersuchen, haben wir uns die angefertigten Klassifikationsbaum-Diagramme aus der Diplomarbeit von Stephan Berndt und Jens Ullrich [11, S.226 ff] für den Anwendungsbereich „Diffraktometrie/Reflektometrie“ vorgenommen.

Ihre entworfenen Diagramme zum Test der neuen Dialogoberflächen sind leicht verständlich und eignen sich aufgrund ihrer Klassifizierung auf den ersten Blick sehr gut zur Attributierung. Dabei wählten wir drei von sechs definierten Diagrammen aus. Es handelt sich um die Diagramme *Dynamische Schrittweite*, *ContinuousScan* und *Einstellungen AreaScan*. Anhand des Diagrammes *Einstellungen AreaScan* soll das Verfahren der Attributierung erläutert werden. Alle vorgestellten Beispiel-Diagramme sind im Verzeichnis `\cte\` des CVS-Moduls `ATOS_XCTLProjekt` zu finden.

1. Klassifikationsbaum „Einstellungen AreaScan“

Obwohl sich das CTE-Diagramm in seiner Form für eine Attributierung eignet, haben wir die Struktur der Klassifikationen ein wenig verändert. Im Diagramm nicht berücksichtigt wurde der Zustand der Theta-Achse. Ein Offset für einen entsprechenden Antrieb lässt sich nur bei nicht fixierter Theta-Achse einstellen.

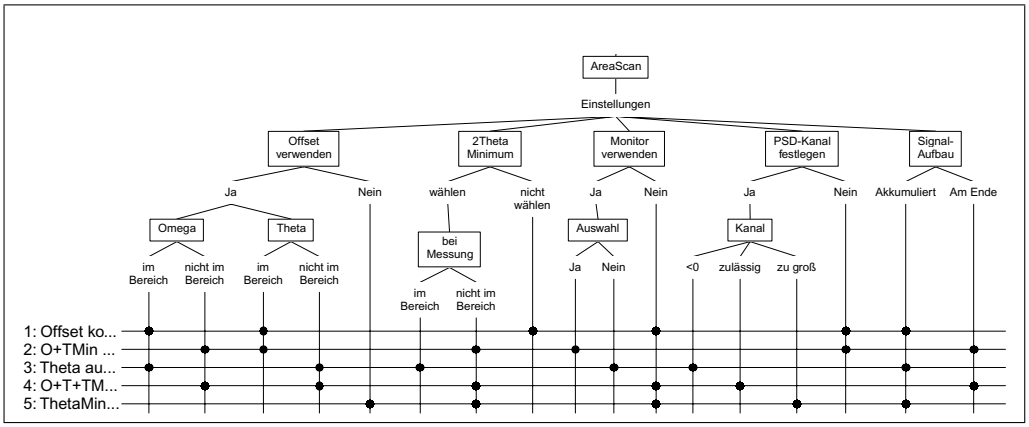


Abbildung 3.12: CTE-Diagramm zum „AreaScan“ (Berndt/Ullrich)

Weiterhin hat sich im Laufe der Projektarbeit die Auswahlfunktion eines Monitordetektors verändert. In der aktuellen Version der Dialogbox zur Festlegung der Scan-Parameter wird nicht mehr über eine Checkbox die Verwendung eines solchen Detektors freigegeben oder unterbunden. Die Auswahl wird vielmehr durch eine bedienungsfreundlichere Combobox realisiert, was die Anpassung des Klassifikationsbaumes notwendig machte.

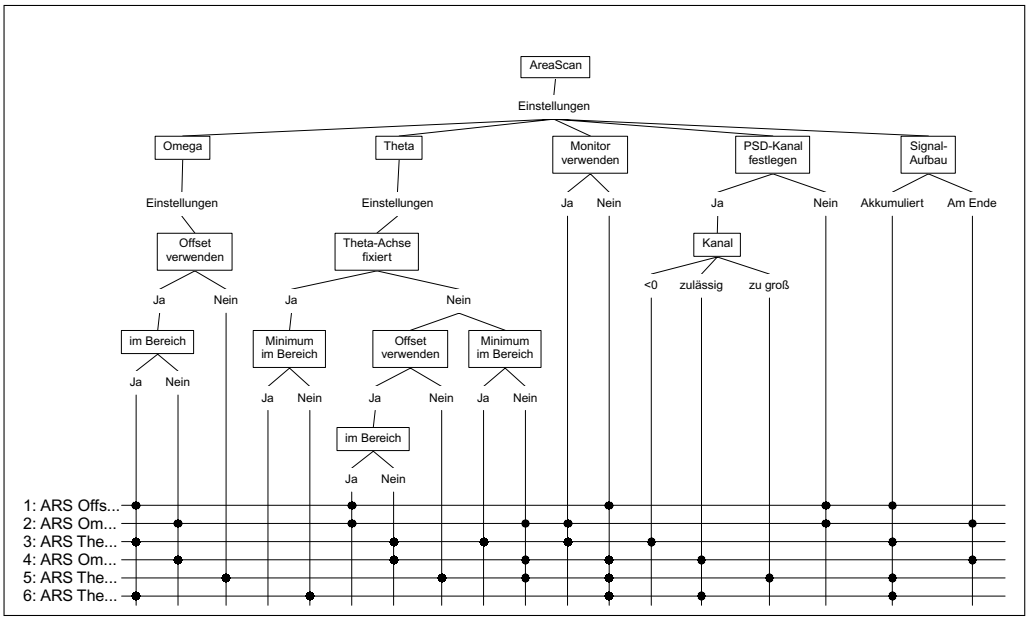


Abbildung 3.13: CTE-Diagramm zum „AreaScan“ (modifiziert)

Die Attributierung eines Klassifikationsbaumes beginnt im Wurzelelement (siehe Abbildung 3.14). Hier werden Skriptkommandos vereinbart, die vor und hinter das gesamte Testskript eines Testfalles gestellt werden sollen (`first`, `last`). Weiterhin werden Variablen zur Identifizierung von Fenstertiteln, Dialogboxen und Konfigurationsdateien vereinbart.

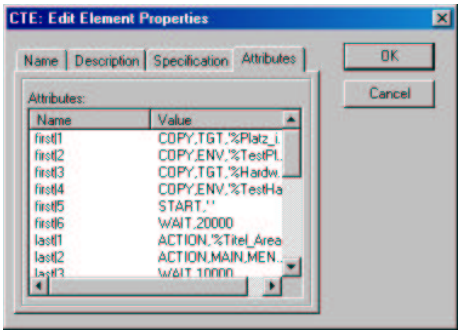


Abbildung 3.14: Attribute des Wurzel-Elements

In den tieferliegenden Elementen werden Skriptkommandos zur Realisierung der ausgewählten Elemente definiert. Je tiefer ein Element im Klassifikationsbaum liegt, desto spezieller sind die spezifizierten Skriptkommandos in seinen Attributen. So werden zum Beispiel in der Klasse „Akkumuliert“ unter der Klassifikation „Signal-Aufbau“ diejenigen Skriptkommandos hinter `step|...` eingetragen, die zur Realisierung der Auswahl dieser Option in der Setup-Dialogbox führen.

Da die Auflistung aller Attribute des CTE-Diagrammes aus Abbildung 3.13 den Rahmen dieses Abschnittes sprengen würde, soll eine Menge ausgewählter Elemente zur Anschauung genügen.

Die Attribute

Die Kommandos im Wurzelelement werden beim Verarbeiten der Testfälle in der Kombinationstabelle vor bzw. hinter jedes erzeugte Testskript geschrieben. Es handelt sich hierbei um Kommandos zum Austauschen der Konfigurationsdateien mit Dateien, die für diesen Testfall präpariert wurden. Die Parameter werden in diesem Fall durch `ini`-Attribute im Element „Einstellungen“ gesetzt.

Element Diffraktometrie

Attribut	Wert
first|1	COPY,TGT,'%Platz_ini',TGT,'%Platz_ini.BAK'
first|2	COPY,ENV,'%TestPlatz_ini',TGT,'%Platz_ini'
first|3	COPY,TGT,'%Hardware_ini',TGT,'%Hardware_ini.BAK'
first|4	COPY,ENV,'%TestHardware_ini',TGT,'%Hardware_ini',FORCE
first|5	START,' '
first|6	WAIT,20000
last|1	ACTION,'%Titel_AreaScan',WINDOW,CLOSE
last|2	ACTION,MAIN,MENU,CLICK,'Datei','Beenden'
last|3	WAIT,10000
last|4	CLEANUP
last|5	COPY,TGT,'%Platz_ini.BAK',TGT,'%Platz_ini'
last|6	DELETE,TGT,'%Platz_ini.BAK'
last|7	COPY,TGT,'%Hardware_ini.BAK',TGT,'%Hardware_ini',FORCE
last|8	DELETE,TGT,'%Hardware_ini.BAK',FORCE
%Titel_AreaScan	Areascan
%Titel_Setup	Einstellungen AreaScan
%Titel_Offset	Einstellungen Offset
%Platz_ini	DEVELOP.INI
%Hardware_ini	HARDWARE.INI
%TestPlatz_ini	%Testname.%Platz_ini
%TestHardware_ini	%Testname.%Hardware_ini

Im Element „AreaScan“ werden Skriptkommandos zum Öffnen und Schließen des AreaScan-Fensters und der Setup-Dialogbox positioniert. Hierbei spielt die Nummer hinter `step|` eine sehr wichtige Rolle. Alle weiteren Skriptkommandos in den Attributen des Klassifikationsbaumes müssen zwischen dem Öffnen und Schließen der Dialogbox liegen, in diesem Fall also zwischen den Schritten 13 und 1000. Für dieses kleine, überschaubare CTE-Diagramm ist das völlig ausreichend. Weiterhin werden noch drei Skriptkommandos an die Positionen 50, 100 und 200 gesetzt. Sie bestimmen, dass die Optionen „Offset verwenden“ und „Theta-Achse fixieren“ standardmäßig nicht gesetzt werden, wenn in den Attributen konstruierender Elemente nichts anderes vereinbart wird. Zusätzlich wird der 1-dimensionale Testdetektor `PSD` als Standard-Detektor ausgewählt.

Element AreaScan

Attribut	Wert
step|10	ACTION,MAIN,MENU,CLICK,'Öffnen','AreaScan-Fenster'
step|11	WINDOWEXISTS,'%Titel_AreaScan',YES
step|12	ACTION,'%Titel_AreaScan',MENU,CLICK,'Setup zum AreaScan...'
step|13	WINDOWEXISTS,'%Titel_Setup',YES
step|50	ACTION,'%Titel_Setup',COMBOBOX,SELECT,'PSD','Detektorauswahl'
step|100	ACTION,'%Titel_Setup',CHECKBOX,UNCHECK,'Offset verwenden'
step|200	ACTION,'%Titel_Setup',CHECKBOX,UNCHECK,'Theta-Achse fixieren'
step|1000	ACTION,'%Titel_Setup',BUTTON,CLICK,'Ok'
step|1100	WINDOWEXISTS,'%Titel_Setup',NO

In den Attributen des Elementes „Einstellungen“ werden die Parameter für die präparierten Konfigurationsdateien vorgenommen. Neben unwesentlichen Angaben zur Fenstergröße und -position werden drei Testdetektoren und zwei Antriebe für Omega und Theta eingerichtet.

Element Einstellungen

Attribut	Wert
ini|%TestPlatz_ini|Steuerprogramm|Environment	Expert
ini|%TestPlatz_ini|Steuerprogramm|xo	30
ini|%TestPlatz_ini|Steuerprogramm|yo	40
ini|%TestPlatz_ini|Steuerprogramm|x1	935
ini|%TestPlatz_ini|Steuerprogramm|y1	725
ini|%TestPlatz_ini|AreaScan|xo	247
ini|%TestPlatz_ini|AreaScan|yo	12
ini|%TestPlatz_ini|AreaScan|x1	628
ini|%TestPlatz_ini|AreaScan|y1	590
ini|%TestPlatz_ini|MOTORSIM|SimulationType	simulation_only
ini|%TestPlatz_ini|MOTORSIM|LogLevel	0
ini|%TestPlatz_ini|MOTORSIM|LogFile	msim.log
ini|%TestPlatz_ini|MOTORSIM|StatusWindow	0
ini|%TestPlatz_ini|MOTORSIM|dll	msim.dll
ini|%TestPlatz_ini|Device0|Name	Counter
ini|%TestPlatz_ini|Device0|Type	Test
ini|%TestPlatz_ini|Device1|Name	Simulant
ini|%TestPlatz_ini|Device1|Type	Simulant
ini|%TestPlatz_ini|Device2|Name	PSD
ini|%TestPlatz_ini|Device2|Type	PSD
ini|%TestPlatz_ini|Motor0|Name	Theta
ini|%TestPlatz_ini|Motor0|Type	C-832
...	...
ini|%TestPlatz_ini|Motor1|Name	Omega
ini|%TestPlatz_ini|Motor1|Type	C-832
...	...

Im Klassen-Element „Ja“ am Ende des Pfades „Omega“ → „Einstellungen“ → „Offset verwenden“ ist eindeutig, dass ein Offset für den Antrieb Omega verwendet werden soll. Entsprechende Skriptkommandos in den Attributen des Elementes sorgen für die Realisierung. Der Schritt 100 aus dem Element „AreaScan“ muss dazu überschrieben werden. Die Schritte 101 und 102 setzen die richtigen Offset-Werte in die Dialogbox. Dabei werden die Variablen `%omega_ist` und `%omega_soll` durch eine tiefer im Baum liegende Klasse definiert.

Da der Offset auch für den Theta-Antrieb definiert werden kann, sollte die Dialogbox zur Einstellung der Offsets erst nach Eintragung dieser Werte geschlossen werden. Zur Realisierung wird die Regel, dass `step`'s mit gleicher Nummer nicht in mehreren konstruierenden Elementen auf verschiedenen Pfaden auftreten dürfen, aufgeweicht. Es ist zulässig, wenn die beiden Skriptkommandos identisch sind. Somit kann die Dialogbox zur Einstellung der Offsets geschlossen werden, wenn entweder nur der Omega- oder nur der

Theta-Offset oder wenn beide Offsets gleichzeitig gesetzt werden. Aus diesem Grund sind die völlig identischen Skriptkommandos mit der Position 110 und 111 in beiden Klassen-Elementen „Ja“ unter „Offset verwenden“ bei Omega und Theta vorzufinden.

Element Ja (Omega-Offset verwenden)

Attribut	Wert
step|100	ACTION,'%Titel_Setup',CHECKBOX,CHECK,'Offset verwenden'
step|101	ACTION,'%Titel_Offset',EDITBOX,EDIT,'%omega_ist','Omega Winkel'
step|102	ACTION,'%Titel_Offset',EDITBOX,EDIT,'%omega_soll','Omega entspricht Winkel'
step|110	ACTION,'%Titel_Offset',BUTTON,CLICK,'OK'
step|111	WINDOWEXISTS,'%Titel_Offset',NO

Ob die Offset-Werte von Omega und Theta für den jeweiligen Testfall in gültigen Bereichen liegen sollen, wird durch Markierung der Klassen-Elemente „Ja“ bzw. „Nein“ unter der Klassifikation „im Bereich“ festgelegt. Dabei müssen nicht nur die beiden Variablen `%omega_ist` und `%omega_soll` entsprechend gesetzt werden, sondern weiterhin muss beachtet werden, welche Auswirkungen ungültige Bereiche haben können.

Sinn dieses Dialogtestes ist zu überprüfen, wie mit falschen Eingabewerten umgegangen wird. Die Dialogbox darf sich bei fehlerhaften Omega- und Thetabereichen nicht schließen. Außerdem werden eine oder mehrere Messageboxen mit einer Fehlermeldung angezeigt. Die Schritte 1100 und 1101 sorgen für die Auswertung dieses Verhaltens, wobei `step|1100` den Schritt aus dem Element „AreaScan“ überschreibt.

Element Ja (Omega-Offset im Bereich)

Attribut	Wert
%omega_ist	5.0
%omega_soll	10.0

Element Nein (Omega-Offset im Bereich)

Attribut	Wert
%omega_ist	5.0
%omega_soll	100.0
step|1100	WINDOWEXISTS,'%Titel_Setup',YES
step|1101	HANDLEMESSAGEBOX,'Meldung',OK,3

Die „Kunst“ bei der Attributierung von zustandsorientierten CTE-Diagrammen liegt in der richtigen Positionierung und Definition von Auswertungsschritten bzw. Skriptkommandos zur Auswertung. So können zum Beispiel drei Eingabefehler in der Dialogbox zu zwei und nicht wie erwartet drei Fehlerbenachrichtigungen über Messageboxen führen, obwohl jeder

Fehler für sich eine solche Benachrichtigung erzeugen würde. Der Testfall-Entwickler muss dieses Verhalten kennen, was die Attributierung zustandsorientierter Klassifikationsbäume äußerst schwierig macht.

Die Testfälle

Ist einmal ein attributierter Klassifikationsbaum fertig gestellt, können beliebige Testfälle durch Markierungen in der Kombinationstabelle konstruiert werden. In unserem Fall haben wir die fünf Testfälle aus der Diplomarbeit von Stephan Berndt und Jens Ullrich um einen weiteren ergänzt, um die Option einer fixierten Theta-Achse zu berücksichtigen.

Testfall 1: Offset korrekt
- Omega-Offset, Messung bleibt im Omega-Bereich
- Theta-Offset, Messung bleibt im Theta-Bereich
- keine Auswahl von 2ThetaMin
- kein Monitor-Detektor
- kein PSD-Kanal festgelegt
- akkumulierte Darstellung

Ergebnis:
- Dialogbox wird geschlossen, Eingabewerte werden übernommen

Testfall 2: Omega- und ThetaMin außerhalb des Bereichs
- Omega-Offset, Messung bleibt nicht im Omega-Bereich
- Theta-Offset, Messung bleibt im Theta-Bereich
- Auswahl von 2ThetaMin, bleibt nicht im Theta-Bereich
- Auswahl eines Monitor-Detektors
- kein PSD-Kanal festgelegt
- Darstellung am Ende einer Messung

Ergebnis:
- Dialogbox wird nicht geschlossen
- Ausgabe von Messageboxen wegen Bereichsüberschreitungen

Testfall 3: Theta außerhalb des Bereichs
- Omega-Offset, Messung bleibt im Omega-Bereich
- Theta-Offset, Messung bleibt nicht im Theta-Bereich
- Auswahl von 2ThetaMin, bleibt im Theta-Bereich
- Auswahl eines Monitor-Detektors
- festgelegter PSD-Kanal ¡ 0
- akkumulierte Darstellung

Ergebnis:
- Dialogbox wird nicht geschlossen
- Ausgabe von Messageboxen wegen Bereichsüberschreitungen
- Ausgabe einer Messagebox mit Angabe der gültigen Kanalnummern des PSD

Testfall 4: Omega, Theta und ThetaMin außerhalb des Bereichs
- Omega-Offset, Messung bleibt nicht im Omega-Bereich
- Theta-Offset, Messung bleibt nicht im Theta-Bereich
- Auswahl von 2ThetaMin, bleibt nicht im Theta-Bereich
- kein Monitor-Detektor
- festgelegter PSD-Kanal zulässig
- Darstellung am Ende der Messung

Ergebnis:
- Dialogbox wird nicht geschlossen
- Ausgabe von Messageboxen wegen Bereichsüberschreitungen

Testfall 5: ThetaMin außerhalb des Bereichs
- Omega-Offset, kein Offset
- Theta-Offset, kein Offset
- Auswahl von 2ThetaMin, bleibt nicht im Theta-Bereich
- kein Monitor-Detektor
- PSD-Kanal zu groß gewählt
- akkumulierte Darstellung

Ergebnis:
- Dialogbox wird nicht geschlossen
- Ausgabe von Messageboxen wegen Bereichsüberschreitungen
- Ausgabe einer Messagebox mit Angabe des Winkelbereichs des PSD

Testfall 6: Theta fixiert und außerhalb des Bereichs
- Omega-Offset, Messung bleibt im Omega-Bereich
- Theta-Offset, kein Offset
- Auswahl von 2ThetaMin, bleibt nicht im Theta-Bereich
- kein Monitor-Detektor
- festgelegter PSD-Kanal zulässig
- akkumulierte Darstellung

Ergebnis:
- Dialogbox wird nicht geschlossen
- Ausgabe einer Messagebox wegen Bereichsüberschreitung

Die Testskripte

Ein attributierter Klassifikationsbaum in Verbindung mit spezifizierten Testfällen bzw. Testsequenzen in der Kombinationstabelle, lässt sich mittels einer Funktion der Testsuite *ATOS* in Testskripte umwandeln und einem Testpaket hinzufügen. Erzeugte Skripte und Konfigurationsdateien werden dafür in die Verzeichnisse \seq\ und \env\ geschrieben.

Das Skript in Abbildung 3.17 zeigt exemplarisch das automatisch generierte Skript für den zweiten Testfall des Klassifikationsbaumes zum „AreaScan“ (vgl. Abbildung 3.13). Weiterhin werden die zwei Konfigurationsdateien `ARS.2.DEVELOP.INI` (Abbildung 3.16) und `ARS.2.HARDWARE.INI` (Abbildung 3.15) angelegt.

```
[MOTORSIM]
SimulationType=simulation_only
LogLevel=0
LogFile=msim.log
StatusWindow=0
dll=msim.dll
[Device0]
Name=Counter
Type=Test
[Device1]
Name=Simulant
Type=Simulant
[Device2]
Name=PSD
Type=PSD
[Motor0]
Name=Theta
Type=C-832
...
[Motor1]
Name=Omega
Type=C-832
...
```

Abbildung 3.15: Hardware-Konfiguration `ARS.2.HARDWARE.INI`

```
[Steuerprogramm]
Environment=Expert
xo=30
yo=40
x1=935
y1=725
[AreaScan]
Environment=Expert
xo=247
yo=12
dx=628
dy=590
```

Abbildung 3.16: Arbeitsplatz-Konfiguration `ARS.2.DEVELOP.INI`

```
COPY,TGT,"DEVELOP.INI",TGT,"DEVELOP.INI.BAK"
COPY,ENV,"ARS.2.DEVELOP.INI",TGT,"‘DEVELOP.INI"
COPY,TGT,"HARDWARE.INI",TGT,"HARDWARE.INI.BAK"
COPY,ENV,"ARS.2.HARDWARE.INI",TGT,"HARDWARE.INI",FORCE
START," "
WAIT,20000
ACTION,MAIN,MENU,CLICK,"Öffnen","AreaScan-Fenster"
WINDOWEXISTS,"Areascan",YES
ACTION,"Areascan",MENU,CLICK,"Setup zum AreaScan..."
WINDOWEXISTS,"Einstellungen AreaScan",YES
ACTION,"Einstellungen AreaScan",COMBOBOX,SELECT,"PSD","Detektorauswahl"
ACTION,"Einstellungen AreaScan",CHECKBOX,CHECK,"Offset verwenden"
ACTION,"Einstellungen Offset",EDITBOX,EDIT,"5.0","Omega Winkel"
ACTION,"Einstellungen Offset",EDITBOX,EDIT,"100.0","Omega entspricht Winkel"
ACTION,"Einstellungen Offset",EDITBOX,EDIT,"5.0","Theta Winkel"
ACTION,"Einstellungen Offset",EDITBOX,EDIT,"10.0","Theta entspricht Winkel"
ACTION,"Einstellungen Offset",BUTTON,CLICK,"OK"
WINDOWEXISTS,"Einstellungen Offset",NO
ACTION,"Einstellungen AreaScan",CHECKBOX,UNCHECK,"Theta-Achse fixieren"
ACTION,"Einstellungen AreaScan",EDITBOX,EDIT,"-15.0","Theta Minimum"
ACTION,"Einstellungen AreaScan",COMBOBOX,SELECT,"Counter","Monitor-Detektor"
ACTION,"Einstellungen AreaScan",CHECKBOX,UNCHECK,"Akkumuliert"
ACTION,"Einstellungen AreaScan",CHECKBOX,CHECK,"Nur am Ende"
ACTION,"Einstellungen AreaScan",BUTTON,CLICK,"Ok"
WINDOWEXISTS,"Einstellungen AreaScan",YES
HANDLEMESSAGEBOX,"Meldung",OK,3
ACTION,"Areascan",WINDOW,CLOSE
ACTION,MAIN,MENU,CLICK,"Datei","Beenden" WAIT,10000 CLEANUP
COPY,TGT,"DEVELOP.INI.BAK",TGT,"DEVELOP.INI"
DELETE,TGT,"DEVELOP.INI.BAK"
COPY,TGT,"HARDWARE.INI.BAK",TGT,"HARDWARE.INI",FORCE
DELETE,TGT,"HARDWARE.INI.BAK",FORCE
```

Abbildung 3.17: Skriptdatei „`ARS Omega- und ThetaMin ausserhalb.HTS`“

2. Klassifikationsbaum „ContinuousScan“

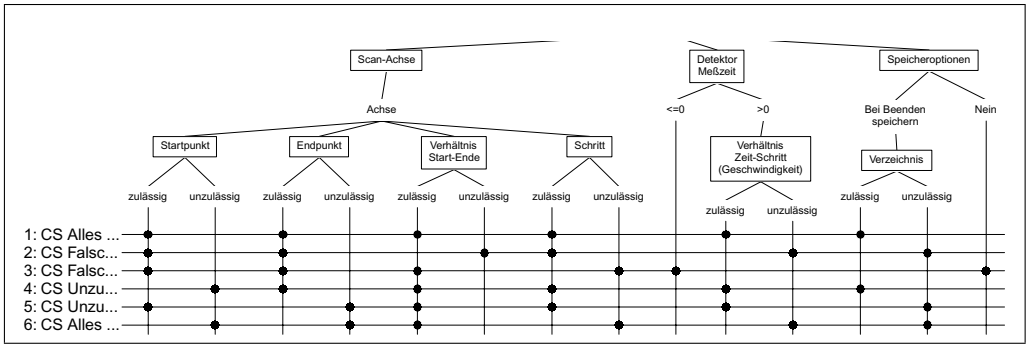


Abbildung 3.18: CTE-Diagramm zum „ContinuousScan“

Die Testfälle

Testfall 1: alle Eingaben korrekt
- zulässiger Startpunkt
- zulässiger Endpunkt
- Startpunkt < Endpunkt
- zulässige Bereichsgröße
- Detektor-Meßzeit > 0 und zulässige Geschwindigkeit
- Bei Beenden Speichern und Speicher-Verzeichnis existiert

Ergebnis:
- Dialogbox wird geschlossen, Eingabewerte werden übernommen

Testfall 2: unzulässige Geschwindigkeit, Start >= Endpunkt
- zulässiger Startpunkt
- zulässiger Endpunkt
- Startpunkt >= Endpunkt
- zulässige Bereichsgröße
- Detektor-Meßzeit > 0, aber unzulässige Geschwindigkeit
- Bei Beenden Speichern und Speicher-Verzeichnis existiert nicht

Ergebnis:
- Dialogbox wird nicht geschlossen
- Hinweis auf ungültige Geschwindigkeit
- Vertauschung von Start- und Endwert
- Verzeichnis-Eintrag auf 'C:\'

Testfall 3: falsche Meßzeit, falscher Schritt (Bereichsgröße)
- zulässiger Startpunkt
- zulässiger Endpunkt
- Startpunkt < Endpunkt
- unzulässige Bereichsgröße
- Detektor-Meßzeit <= 0
- kein Speichern
  nicht

Ergebnis:
- Dialogbox wird nicht geschlossen
- Bereichsgröße auf minimal erlaubte (Motorschrittweite) gesetzt
- Meßzeit auf 1.0 gesetzt

Testfall 4: unzulässiger Start
- unzulässiger Startpunkt
- zulässiger Endpunkt
- Startpunkt < Endpunkt
- zulässige Bereichsgröße
- Detektor-Meßzeit > 0 und zulässige Geschwindigkeit
- Bei Beenden speichern und Speicher-Verzeichnis existiert

Ergebnis:
- Dialogbox wird nicht geschlossen
- Startpunkt wird auf Minimalwert + Schrittweite des Motors gesetzt
- Messagebox zu fehlerhaftem Startwert

Testfall 5: unzulässiges Ende
- zulässiger Startpunkt
- unzulässiger Endpunkt
- Startpunkt < Endpunkt
- zulässige Bereichsgröße
- Detektor-Meßzeit > 0 und zulässige Geschwindigkeit
- Bei Beenden speichern und Speicher-Verzeichnis existiert nicht

Ergebnis:
- Dialogbox wird nicht geschlossen
- Endwert wird auf Maximalwert - Schrittweite des Motors gesetzt
- Messagebox zu fehlerhaftem Endwert

3. Klassifikationsbaum „Dynamische Schrittweite“

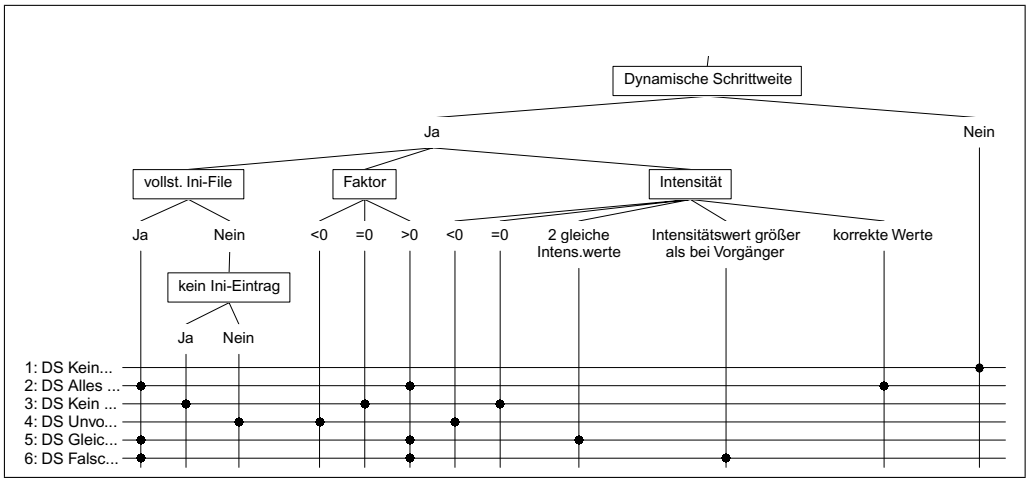


Abbildung 3.19: CTE-Diagramm zur „Dynamische Schrittweite“

Die Testfälle

Testfall 1: Dynamische Schrittweite aus
- Dynamische Schrittweite aus

Ergebnis:
- Messung ohne dynamischer Schrittweiten-Steuerung kann gestartet werden

Testfall 2: alle Eingaben korrekt
- Dynamische Schrittweite an
- vollständige Einträge im Ini-File
- alle Faktorwerte sind größer 0
- alle Intensitätswerte sind korrekt

Ergebnis:
- Messung mit dynamischer Schrittweiten-Steuerung kann gestartet werden

Testfall 3: kein Ini-File und Faktor- und Intensitätswerte gleich 0
- Dynamische Schrittweite an
- fehlende Einträge im Ini-File
- ein Faktorwert gleich 0
- ein Intensitätswert gleich 0

Ergebnis:
- Dialogbox enthält bei Aufruf Standardwerte, alle Faktorwerte= 1.0, alle Intensitätswerte in 1000-er Schritten, beginnend bei 0
- Dialogbox wird nicht geschlossen, falsche Faktorwerte automatisch auf 1.0 gesetzt, falsche Intensitätswerte an das Ende mit nächstmöglichem Wert (um 1 größer als aktuelles Maximum) und zugehöriger Faktorwert auf 1.0 gesetzt

Testfall 4: unvollständiges Ini-File, Faktor- und Intensitätswerte < 0
- Dynamische Schrittweite an
- unvollständige Einträge im Ini-File
- ein Faktorwert kleiner 0
- ein Intensitätswert kleiner 0

Ergebnis:
- Dialogbox enthält bei Aufruf teilweise (ab fehlendem Eintrag) Standardwerte, entsprechende Faktorwerte = 1.0, alle Standard-Intensitätswerte in 1000-er Schritten
- Dialogbox wird nicht geschlossen, falsche Faktorwerte automatisch auf 1.0 gesetzt, falsche Intensitätswerte an das Ende mit nächstmöglichem Wert (um 1 größer als aktuelles Maximum) und zugehöriger Faktorwert auf 1.0 gesetzt

Testfall 5: gleiche Intensitätswerte
- Dynamische Schrittweite an
- unvollständige Einträge im Ini-File
- alle Faktorwerte sind größer 0
- 2 gleiche Intensitätswerte

Ergebnis:
- Dialogbox enthält bei Aufruf vollständig die Ini-Einträge
- Dialogbox wird nicht geschlossen, der zweite gleiche Intensitätswert wird an das Ende mit nächstmöglichem Wert (um 1 größer als aktuelles Maximum) und zugehöriger Faktorwert auf 1.0 gesetzt

Testfall 6: falsche Reihenfolge der Intensitätswerte
- Dynamische Schrittweite an
- vollständige Einträge im Ini-File
- alle Faktorwerte sind größer 0
- ein Intensitätswert ist größer als sein Nachfolger

Ergebnis:
- Dialogbox enthält bei Aufruf vollständig die Ini-Einträge
- Dialogbox wird nicht geschlossen, die Einträge für die Intensitäten werden sortiert, die Faktor-Einträge entsprechend mit umsortiert

Beispiel für aktionsorientierte Diagramme

Die Möglichkeit zur Attributierung aktionsorientierter CTE-Diagramme soll anhand eines Beispiels zum Anwendungsbereich „LineScan“ demonstriert werden. Ausgangspunkt bildet ein Klassifikationsbaum (siehe Abbildung 3.20), der aufgrund seiner Komplexität durch sogenannte „Refinement“-Elemente in viele Unterdiagramme strukturiert ist.

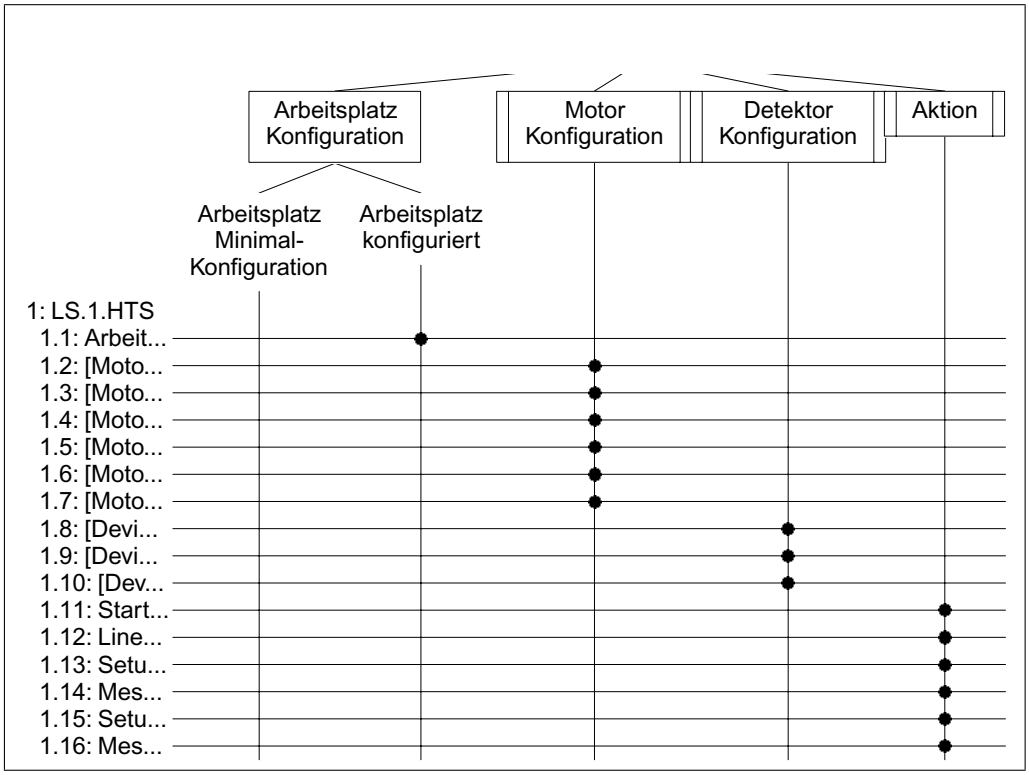


Abbildung 3.20: CTE-Diagramm zum „LineScan“

Die folgenden Abbildungen zeigen eine Auswahl von insgesamt 13 verschachtelten Klassifikationsbäumen, um an ihnen die prinzipielle Struktur des CTE-Diagramms zu erläutern. Das vollständige Diagramm (ohne Testsequenzen) kann im Anhang D eingesehen werden.

Bei der Konstruktion eines geeigneten Klassifikationsbaumes zur Unterstützung beim systematischen Testfallentwurf wurde besonders die Bedeutung der Konfigurationsdateien für das Steuerungsprogramm berücksichtigt. In den ersten Testschritten jeder Testsequenz ist es möglich, in den Diagrammen eine Vielzahl möglicher Konfigurationszustände festzulegen. Die Abbildungen 3.21 und 3.22 zeigen dies am Beispiel der Motorparameter. Ein äquivalentes Diagramm dient zur Bestimmung der Detektorparameter.

Die Ergebnisse aus Abschnitt 3.6, in dem die Bedeutungen der einzelnen Hardware-Parameter innerhalb der Umgebungssimulation untersucht wurden, waren ausschlaggebend bei der Spezifikation von Klassifikationen für die Ini-Parameter. Dabei wurden nur solche Parameter berücksichtigt, die in irgend einer Weise für das Verhalten eines hardwareunabhängigen Testgerätes verantwortlich sind. Parameter, die auch für Testgeräte festgelegt werden können, aber zu keinen Auswirkungen innerhalb der Umgebungssimulation führen, sind besonders gekennzeichnet.

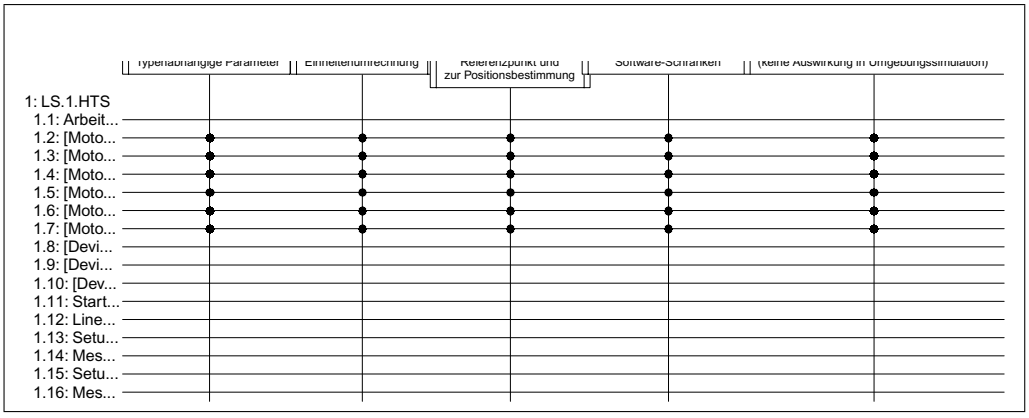


Abbildung 3.21: CTE-Diagramm zum „LineScan“ (Motorparameter)

Die Abbildung 3.23 zeigt den wichtigsten Klassifikationsbaum, in dem die tatsächlichen Testsequenzen durch Markierungen von Aktions-Klassen gebildet werden. Das Diagramm ist so aufgebaut, dass es unmöglich ist, in einem Testschritt mehrere Aktionen in verschiedenen Dialogboxen oder Fenstern gleichzeitig vorzunehmen. Dafür sorgt die übergeordnete Klassifikation

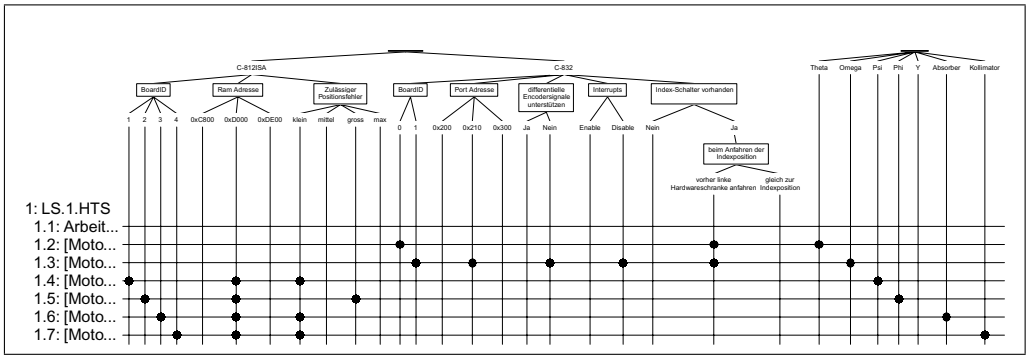


Abbildung 3.22: CTE-Diagramm zum „LineScan“ (Motorparameter - Typen)

„Umgebung“. In aufeinanderfolgenden Testschritten gibt es keine Einschränkung, welche Aktion auf welches Ziel ausgeführt werden darf. Das verlangt ein diszipliniertes Vorgehen des Testfall-Entwicklers. Er muss die Reaktionen des XCTL-Systems auf bestimmte Aktionen kennen. Es ist zum Beispiel absurd, Eingaben in eine File-Dialogbox vorzunehmen, die durch keinen vorherigen Schritt zum Aufruf veranlasst wurde.

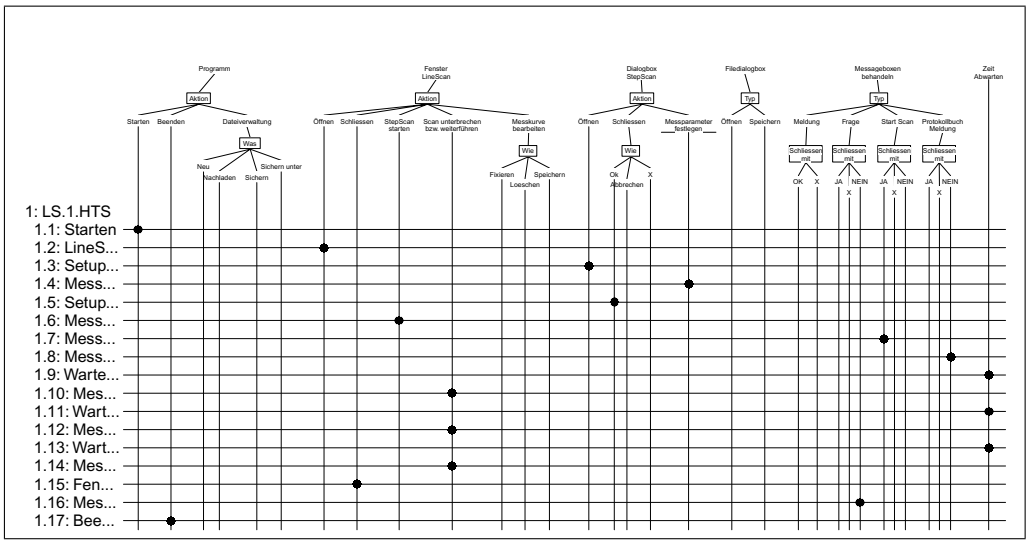


Abbildung 3.23: CTE-Diagramm zum „LineScan“ (Aktionen)

Das stellt ein größeres Problem dar, als zunächst angenommen. Das Verhalten des Steuerprogrammes ist von vielen Parametern abhängig, die sich gegenseitig beeinflussen. So dauert zum Beispiel die Messung eines bestimmten Bereichs länger, wenn für den zuständigen Antrieb in den Ini-Parametern ein niedrigerer Geschwindigkeitswert festgelegt wird. Der nächste Testschritt darf erst nach einer bestimmten Wartezeit erfolgen, die zur Zeit der Konstruktion einer Testsequenz noch nicht bekannt ist. Einige Kombinationen von Parametern könnten auch zu Fehlermeldungen des Steuerprogrammes noch vor der zu startenden Messung führen.

Alle diese Verhaltensweisen des XCTL-Systems müssen dem Testfall-Entwickler bekannt sein. Der Entwurf einer Testsequenz im Klassifikationsbaum muss demzufolge schrittweise erfolgen. Referenzdateien, die zum Vergleich mit Ausgabedateien oder Kurvendateien herangezogen werden sollen, müssen zunächst erstellt werden. Meßzeiten müssen eventuell ermittelt werden, was wiederum die erstmalige Ausführung der konstruierten Testsequenz erfordert. Mit diesen Daten kann die Testsequenz schließlich vervollständigt werden.

Das Verfahren der Klassifikationsbaum-Methode mit aktionsorientierten Diagrammen kann dem Entwickler von Testskripten für das Testsystem *ATOS* dennoch Unterstützung bieten. Viele nahezu identische Skripte können mit wenigen Modifikationen sehr schnell und übersichtlich durch „Copy&Paste“ erstellt werden. Alle Konfigurationsdateien und Testskripte liegen in einer einzigen Datei und können bei Bedarf aus dem Diagramm heraus extrahiert werden. Eine vollautomatische Gewinnung von Testsequenzen durch systematische Markierung aller Punkte in der Kombinationstabelle ist aufgrund der Abhängigkeiten von aufeinanderfolgenden Test-Schritten und der Eingabedaten untereinander für komplexe Anwendungsfälle wie den „LineScan“ aber nicht möglich.

3.8.2 Unterstützte Kommando-Konstruktion

Der manuelle Entwurf von HTS-Testsequenzen mit einem beliebigen Texteditor gestaltet sich trotz einer benutzerfreundlichen Kommando-Syntax nicht immer einfach. Schnell können sich Fehler einschleichen, die eine Ausführung der Skripte unmöglich machen. Der Entwickler von Testskripten muss alle möglichen Bezeichner von Fenstern, Dialogboxen, Menüs und Steuerelementen aus der Ressourcen-Datenbank eines Testobjektes genauestens kennen. Einige Kommandos besitzen optionale Parameter, deren Bedeutungen erst beim Studieren der Skiptsprachen-Spezifikation ersichtlich werden.

Aus diesen Gründen entstand der Wunsch nach Funktionalitäten zur Unterstützung bei der Konstruktion von gültigen und konsistenten Skriptkommandos, die sich in den Funktionspunkten `\F40\`, `\F50\` und `\F60\` des Pflichtenheftes von *ATOS* aus Abschnitt 2.7 niederschlagen. Zur Erstellung und Bearbeitung eines Testschrittes bietet *ATOS* alle Skriptkommandos kommentiert in einer Dialogbox zur Auswahl an.

Die Möglichkeiten zur Definition eines Parameters ist sowohl vom aktuellen Skriptkommando, als auch von vorangegangenen Parametern abhängig. *ATOS* stehen zur Zeit seiner Ausführung Listen mit allen Oberflächen-Elementen des Testobjektes zur Verfügung (Ressourcen-Datenbank), die dem Entwickler von Testfällen an den jeweiligen Parameterpositionen zur Auswahl angeboten werden. Wie in den Abbildungen 3.24 und 3.25 zu erkennen, gestaltet sich die Konstruktion eines vollständigen HTS-Kommandos in mehreren Schritten mit einer sich dynamisch anpassenden Dialogbox.

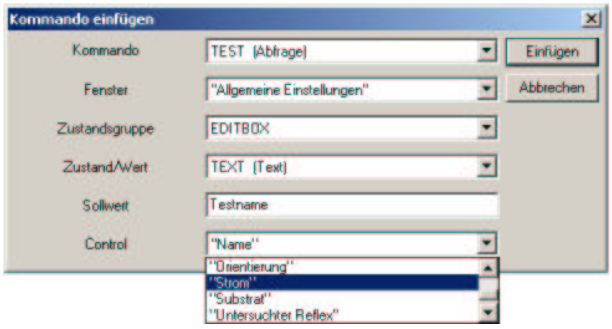


Abbildung 3.24: Konstruktion eines `TEST`-Kommandos mit *ATOS*

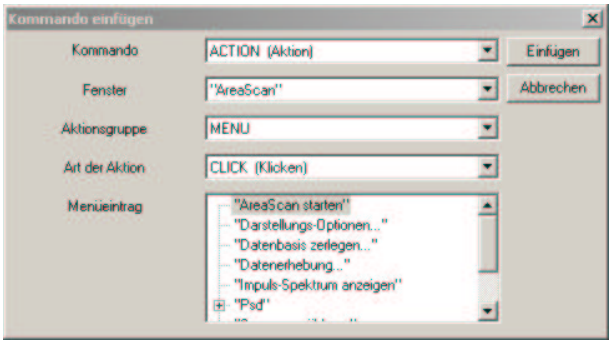


Abbildung 3.25: Konstruktion eines `ACTION`-Kommandos mit *ATOS*

Skriptkommandos der höheren Sprache HTS werden wie im Abschnitt 3.5.1 beschrieben über die Regeldatei `hts2ats.rul` in die tiefere Skriptsprache ATS übersetzt. Auf diese Weise kann der Sprachumfang sehr leicht erweitert und angepasst werden. Auch die ATOS-Komponente zur Unterstützung bei der Kommandokonstruktion muss über Veränderungen an der Sprachspezifikation Bescheid wissen, weshalb neben `hts2ats.rul` eine weitere Regeldatei (Abbildung 3.26) zur Definition der Syntax gehalten wird. Hier wird jedes Skriptkommando mit seinen Parametern und Kommentaren in entsprechende Regeln für den Matcher (siehe Kapitel 4 und Anhang A) formuliert.

```
DESCRIPTION,"HTS-Command-Specification v24 010902"
BEGINSTATE,"NORMAL"

# Gültige Kommandos mit Beschreibung
#
RULE
   STATE    "NORMAL"
   PATTERN  "GETCOMMANDS"
   OUTPUT   "COMMANDS","ACTION","COPY",          "CLEANUP",  "COMPARE",  "DELETE" ...
   OUTPUT   "DESC",    "Aktion","Datei kopieren","Aufräumen","Vergleich","Datei löschen" ...
   NEWSTATE "NORMAL"
ENDRULE

# "Unsichtbare" Kommandos
#
RULE
   STATE    "NORMAL"
   PATTERN  "HIDDEN"
   OUTPUT   "ENDLOOP"
   NEWSTATE "NORMAL"
ENDRULE

# Verknüpfungen der Kommandos
#
RULE
   STATE    "NORMAL"
   PATTERN  "LINK","LOOP"
   OUTPUT   "after","ENDLOOP"
   NEWSTATE "NORMAL"
ENDRULE
RULE
   STATE    "NORMAL"
   PATTERN  "LINK","ENDLOOP"
   OUTPUT   "before","LOOP"
   NEWSTATE "NORMAL"
ENDRULE

# Interaktive Kommandos
#
RULE
   STATE    "NORMAL"
   PATTERN  "INTERACTIVE"
   OUTPUT   "QUESTION","MESSAGE"
   NEWSTATE "NORMAL"
ENDRULE

# Kommando ACTION
#
RULE
   STATE    "NORMAL"
   PATTERN  "GETPARAM","ACTION"
   OUTPUT   "IPARAM","window"
   OUTPUT   "DESC","Fenster"
   NEWSTATE "NORMAL"
ENDRULE
...
```

Abbildung 3.26: Regeldatei `hts.syn` zur Beschreibung der HTS-Syntax

Zusätzlich lässt sich mit den Regeln festlegen, ob das Kommando einen interaktiven Eingriff vom Tester benötigt. Skripte die solche Kommandos wie `MESSAGE` und `QUESTION` enthalten werden von *ATOS* erkannt und lassen sich an den Anfang der durchzuführenden Testsequenzen einsortieren. Im Benutzerleitfaden von *ATOS* [19] wird der Umgang mit Regeldateien zur Spezifizierung der Sprachsyntax ausführlich beschrieben.

Kapitel 4

Das Design

4.1 Klassenbeziehungen

Das Design unseres Testsystems hat sich entgegen der üblichen Softwareentwicklungsphasen erst während der Implementierung herausgebildet. Für jeden abgrenzbaren Problem- bzw. Aufgabenbereich wurde eine neue Komponente (C++ Klasse) entworfen, die so weit wie möglich auf bereits entwickelte Funktionalitäten anderer Klassen zugreift. Einige Aufgaben sind erst zur Zeit der Bearbeitung des Themas aus Ideen zur Verbesserung und Erweiterung des Testsystems entstanden und waren zu Beginn der Studienarbeit noch nicht abzusehen.

Offensichtlich benötigt man grundlegende Funktionalitäten zum Einlesen und Bearbeiten strukturierter ASCII-Dateien wie Testskripte und RC-Dateien in den Speicher. Dafür sind schon früh die Klassen `LineParser` und `Matcher` implementiert worden. Im Zuge der Verwaltung von Testsequenzen mit *ATOS* entstanden zusätzlich die Klassen `File`, `FileSet`, und `TestSequence`. Zur Verwaltung von Strukturinformationen, Zeilen und Wörtern aus den Textdateien dienen die Hilfs-Klassen `Line`, `LineList`, `LineListEntry`, `Entry`, `LLERef`, `LLERefList` und `VariableSet`.

Diese Klassen werden extensiv von den Parser-Klassen verwendet. Eine Methode der Klasse `RCParser` liest Ressourcen (RC-Dateien) ein und wandelt sie mit Funktionen der Klasse `Matcher` und einer Regeldatei in eine Ressourcen-Datenbank (URF-Datei) um. Die Klasse `URFParser` liest mit einer Methode diese Datenbank in den Speicher und stellt mit weiteren Methoden Zugriffe auf die Projekt-Ressourcen für andere Komponenten zur Verfügung. Die Parser-Klassen `ATSParser` und `HTSParser` sind letztendlich für die eigentliche Ausführung von Skriptkommandos verantwortlich.

Etwas abseits davon stehen Klassen, die für den Dateivergleich, die Testsequenzerzeugung und die Verwaltung der Projektressourcen implementiert wurden. Das betrifft die Klassen `DataParser`, `Toleranz`, `Toleranzen`, `CTEParser` `Element`, `IDList`, `Teststeps`, `CommandProcessor`, `RCParser` und `RCEditor`.

Die Klasse `ATOS` stellt alle Funktionalitäten der Testsuite zur Verfügung. Bis auf die Klassen `DataParser`, `Toleranz`, `Toleranzen`, `RCParser` und `RCEditor`, die ausschließlich zur Bildung des Dateivergleichers *DataDiff* und des Ressourcenverwalters *RCParser* benötigt werden, greift `ATOS` dazu direkt oder indirekt auf Funktionen aller anderen Klassen zu. Das Design der Testsuite lässt sich wie in Abbildung 4.1 dargestellt in drei Schichten unterteilen. Die grafische Benutzerschnittstelle und ein unabhängiger Prozeß zur Ausführung von Testskripten kommunizieren miteinander und bilden die oberste Schicht des Modelles.

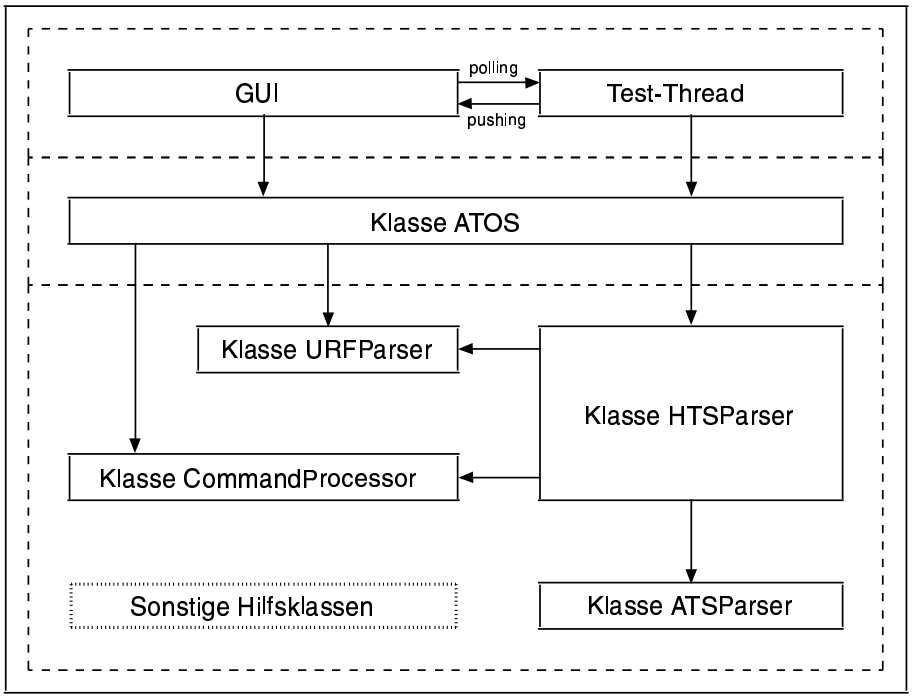


Abbildung 4.1: 3-Schichten-Modell der Testsuite *ATOS*

Die Klasse `ATOS` hat als Vermittler zwei Aufgaben. Einerseits greift sie massiv auf Funktionalitäten der Parser- und Hilfsklassen aus der untersten Schicht zu. Da alle benutzten Klassen verborgen werden, müssen GUI und Test-Thread nur noch auf die Schnittstellen einer Instanz der Klasse `ATOS` zugreifen. Andererseits stellt sie Funktionalitäten zur Verwaltung von Testsequenzen und Generierung von Skriptkommandos bereit. Bei der Konstruktion von neuen Kommandos wird die korrekte Syntax mit Hilfe der Klasse `CommandProcessor` und die Konsistenz der benutzen Oberflächen-Elemente mittels Funktionalitäten der Klasse `URFParser` sichergestellt. Syntax und Konsistenz ist ebenso vor der Ausführung bereits vorhandener HTS-Skripte sicherzustellen, weshalb die Klasse `HTSParser` ebenfalls auf Funktionen der beiden Klassen `CommandProcessor` und `URFParser` zugreifen muss.

Um einen Eindruck von den komplexen Beziehungen zwischen den Klassen des Testsystems und seiner Komponenten zu bekommen, dient die Abbildungen 4.2 eines UML-Klassendiagrammes, welches mit Hilfe des CASE-Tools *Together* angefertigt wurde. Zu beachten ist, dass ein Klassendiagramme nur statische Beziehungen darstellen. Dynamische Instanzzierungen werden bei der Generierung eines UML-Klassendiagrammes mit *Together* nicht berücksichtigt. So ist beispielsweise nicht sichtbar, dass Objekte der Klasse `LineParser` in jeder Instanz eines Parsers wie `ATSParser`, `HTSParser` oder `URFParser` temporär angelegt und benutzt werden.

Statische Beziehungen zwischen Klassen bestehen, wenn ihre Memberattribute Instanzen oder Zeiger auf Klassen sind, oder wenn sie in einer Verebungsrelation zueinander stehen. Dazu werden nur die Klassendefinitionen in den Headerdateien und nicht die Methodendefinitionen in den `.CPP`-Quelltextdateien untersucht. Aus diesem Grund steht die Klasse `RCParser` scheinbar völlig isoliert im UML-Klassendiagramm (unten links), obwohl sie offenbar Objekte der Klassen `LineParser` und `Matcher` zwischenzeitlich in ihren Methoden verwendet. Mit der Abbildung 4.2 kann jedoch sehr gut verdeutlicht werden, dass die Hilfsklassen eine wesentliche Rolle für das Design unserer Programmes spielen und in jeder Komponente ihren Einsatz finden.

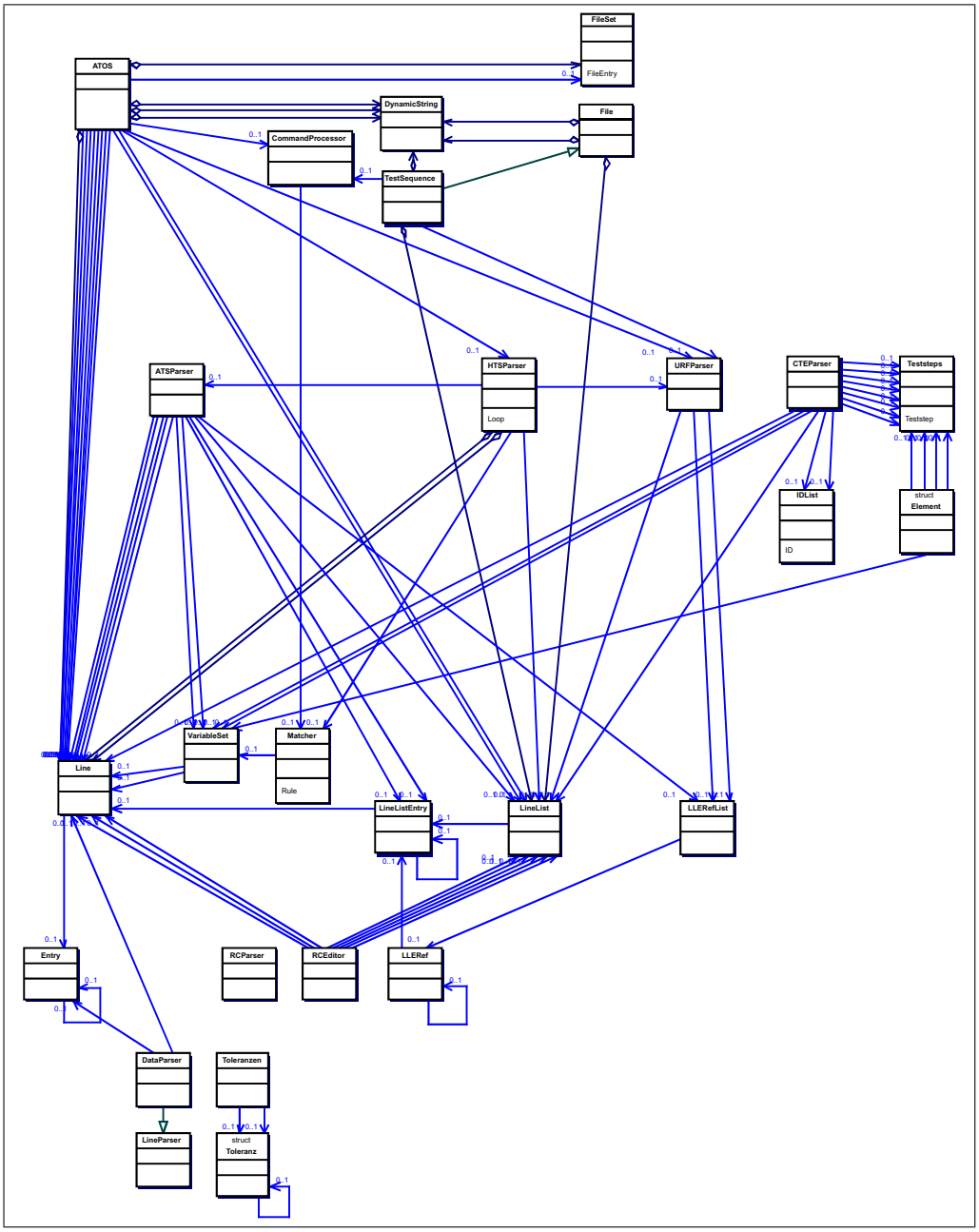


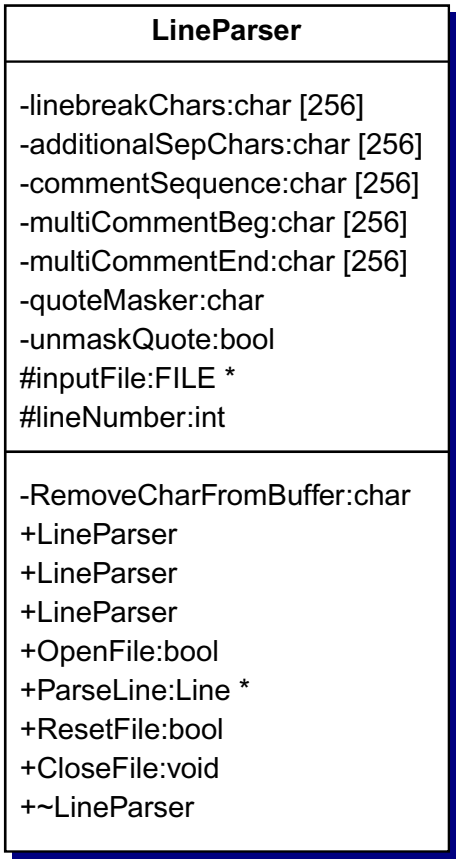
Abbildung 4.2: UML-Klassendiagramm der Testsystem-Komponenten (Stand vom 01.10.2002)

4.2 Beschreibung wesentlicher Klassen

4.2.1 Komponente *ATOS*

Klasse `LineParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse bietet die Möglichkeit eine ASCII-Datei einzulesen und ihren Inhalt in logische Zeilen zu zerlegen. Über die Konstruktoren kann die Zerlegung konfiguriert werden. Somit lässt sich das Verhalten dieser Klasse an vielfältige Dateitypen anpassen.

3. Beschreibung wesentlicher Methoden

`LineParser(const char* linebreakCh, const char* addSepChars, const char* commentSeq, const char* commentBeg, const char* commentEnd, const char maskQuote, bool translateQuote)`
Über die Parameter des Konstruktors kann das Verhalten der Klasse variiert werden.

`linebreakCh:` Definition von Zeichen, die am Ende einer physischen Zeile stehend die aktuelle logische Zeile noch nicht abschließen. (meistens Trennzeichen)

`addSepChars:` Definition von Trennzeichen zwischen Einträgen (Leerzeichen und Komma default)

`commentBeg:` Definition einer Zeichensequenz zur Kennzeichnung des Beginns mehrzeiliger Kommentare (z.B. /*)

`commentEnd:` Definition einer Zeichensequenz zur Kennzeichnung des Endes mehrzeiliger Kommentare (z.B. */)

`commentSeq:` Definition einer Zeichensequenz zur Kennzeichnung eine einzeiligen Kommentars (z.b. //)

`maskQuote:` Definition eines Zeichens zur Kennzeichnung von Anführungszeichen in Strings (von " umschlossene Zeichenketten).

`translateQuote:` Ist dieser Parameter `true`, werden alle in Strings maskierten Anführungszeichen ohne das Maskierungszeichen eingelesen.

`bool OpenFile(const char* fname)` Öffnet die zu parsende Textdatei

`fname:` Dieser Parameter gibt den Pfad und den Namen der Datei an, die geöffnet werden soll.

`Rückgabewert: true`, wenn die Datei erfolgreich geöffnet werden konnte.

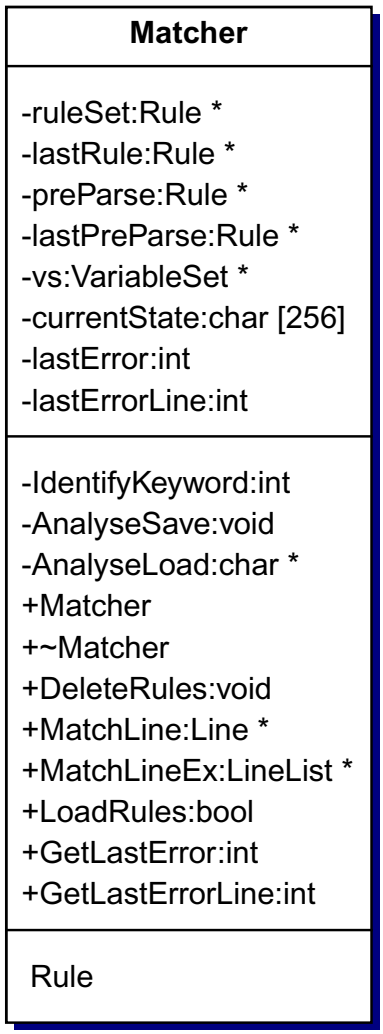
`Line* ParseLine(void)` Liest eine oder mehrere physische Zeilen aus der geöffneten Datei und erzeugt daraus eine logische Zeile in Form eines `Line`-Objektes.

`Rückgabewert:` Bei erfolgreichem Lesen wird ein Zeiger auf ein `Line`-Objekt zurückgegeben, das die logische Zeile repräsentiert.

`void CloseFile(void)` Schließt die aktuell geöffnete Datei.

Klasse `Matcher`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse bietet die Möglichkeit eine logische Zeile eines `Line`-Objektes mit einem Satz von Regeln zu vergleichen. Der Regelsatz wird dazu in einer externen Textdatei formuliert und von der Klasse `Matcher` bei Bedarf in den Speicher eingelesen. Jede Regel muss einer festgelegten Syntax entsprechen, um vom Matcher verarbeitet zu werden. Die grundsätzliche Arbeitsweise des

Matchers entspricht der einer Zustandsmachine. Das Pattern und der aktuelle Zustand bestimmen die Ausgabezeilen (`Line`-Objekte) zur Aufnahme in ein `LineList`-Objekt und einen neuen Zustand.

Die folgenden Regeln führen zu einem Übergang von `ZUSTAND1` in `ZUSTAND2`, wenn die vorliegende logische Zeile aus einem Wort (`Entry`-Objekt) „z1“ besteht. Zudem wird eine logische Zeile mit den zwei Wörtern „Zustand1“ und „Zeile1“ erstellt. Trifft der Matcher im `ZUSTAND2` auf eine logische Zeile mit „z2“ als erstes Wort, wird in `ZUSTAND1` zurück gewechselt. Außerdem wird das zweite Wort in `variable` abgespeichert und beim Ausschreiben der zwei Zeilen hinter „Zustand2“, „Zeile2“ eingefügt. Ein ausführliches Dokument zum Umgang mit Regeldateien für den Matcher ist im Anhang A zu finden.

```
BEGINSTATE  "ZUSTAND1"

RULE
   STATE    "ZUSTAND1"
   PATTERN  "z1"
   OUTPUT   "Zustand1","Zeile1"
   NEWSTATE "ZUSTAND2"
ENDRULE

RULE
   STATE    "ZUSTAND2"
   PATTERN  "z2",<SAVE: variable>
   OUTPUT   "Zustand2","Zeile1"
   OUTPUT   "Zustand2","Zeile2",<LOAD: variable>
   NEWSTATE "ZUSTAND1"
ENDRULE
```

3. Beschreibung wesentlicher Methoden

`LineList* MatchLineEx(Line* lineIn)` Versucht eine Zeile mit dem vorhandenen Regelsatz zu matchen und erzeugt ggf. eine Liste von Ausgabezeilen.

`lineIn:` Dieser Parameter zeigt auf ein `Line`-Objekt, das mit dem aktuellen Regelsatz verglichen werden soll.

`Rückgabewert:` Wenn eine Regel matched, wird ein Zeiger auf ein `LineList`-Objekt zurückgegeben. Wurde keine passende Regel gefunden, wird 0 geliefert.

`bool LoadRules(const char* file)` Liest einen Satz von Regeln aus einer Datei. Bereits vorhandene Regeln werden überschrieben bzw. gelöscht.

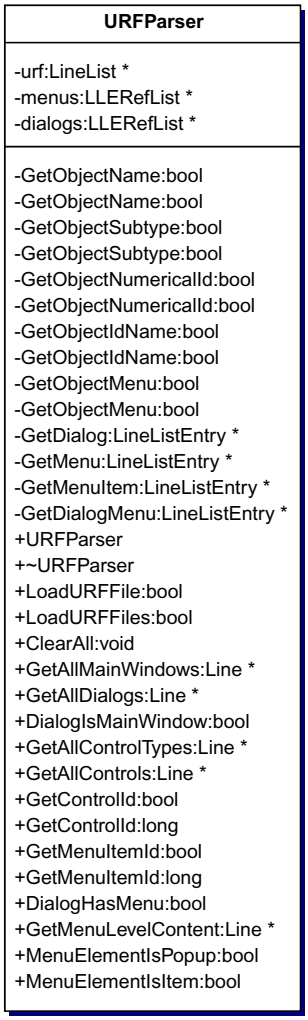
`file:` Gibt den Namen (incl. Pfad) der Regeldatei an.

`Rückgabewert:` Bei erfolgreichem Laden, wird `true` zurückgegeben.

`void DeleteRules(void)` Löscht alle geladenen Regeln.

Klasse `URFParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse ermöglicht den Zugriff auf Informationen über alle Oberflächen-Elemente einer Windows-Anwendung aus einer oder mehreren Ressourcen-Datenbanken (URF-Dateien).

3. Beschreibung wesentlicher Methoden

`bool LoadURFFiles(Line* files)` Lädt URF-Datei(en) in den Speicher. Jeder Aufruf fügt die Daten den bereits geladenen hinzu.

`files:` Liste von URF-Dateien (incl. Pfaden) in Form eines `Line`-Objektes. Somit können mehrere Dateien auf einmal geladen werden.

`Rückgabewert:` Wenn die Datei(en) erfolgreich geladen wurde(n), wird `true` zurückgegeben.

`void ClearAll(void)` Löscht alle bisher geladenen Daten.

`Line* GetAllMainWindows(void)` Ermittelt alle Dialogboxen, die als Hauptfenster markiert sind.

`Rückgabewert:` Wenn Dialogboxen bzw. Fenster vorhanden sind, die als Hauptfenster markiert sind, wird ein Zeiger auf ein `Line`-Objekt geliefert, dessen Einträge die Namen der Hauptfenster beinhalten.

`Line* GetAllDialogs(bool excludeMainWindows)` Ermittelt alle Dialogboxen, die in der geladenen Ressourcen-Datenbank vorhanden sind.

`excludeMainWindows:` Wenn dieser Parameter `true` ist, werden die Fenster bzw. Dialogboxen, die als Hauptfenster markiert sind nicht berücksichtigt. Anderenfalls werden alle Fenster bzw. Dialogboxen ermittelt.

`Rückgabewert:` Wenn Dialogboxen vorhanden sind, wird ein Zeiger auf ein `Line`-Objekt geliefert, dessen Einträge die Bezeichnungen der Dialogboxen beinhalten.

`Line* GetAllControls(const char* dialog, const char* controlType)` Liefert alle Controls einer Dialogbox.

`dialog:` Bezeichnung der Dialogbox, die durchsucht werden soll.

`controlType`: Bezeichnung eines Typs, zur Einschränkung der ermittelten Controls. Bei 0 werden alle Controls ermittelt (doppelte Einträge möglich, weil gleiche Bezeichner für unterschiedliche Controltypen erlaubt sind).

`Rückgabewert`: Wenn Controls (mit dem angegebenen Typ) innerhalb der Dialogbox existieren, wird ein Zeiger auf ein `Line`-Objekt geliefert, dessen Einträge die Bezeichnungen der gefundenen Controls beinhalten.

`bool GetControlId(const char* dialog, const char* controlType, const char* controlName, char* buffer, unsigned int size)`

Ermittelt die numerische Id eines Controls von einem bestimmten Typ innerhalb einer Dialogbox.

`dialog`: Bezeichnung der Dialogbox, die durchsucht werden soll.

`controlType`: Typ des Controls, dessen numerische Id ermittelt werden soll.

`controlName`: Bezeichner des Controls, dessen numerische Id ermittelt werden soll.

`buffer`: Zeiger auf einen Puffer, in dem die numerische Id als Zeichenkette gespeichert werden soll.

`size`: Größe des Puffers.

`Rückgabewert`: Bei Erfolg wird `true` geliefert und im Puffer befindet sich die numerische Id als Zeichenkette.

`bool GetMenuItemId(const char* dialog, Line* path, char* buffer, unsigned int size)`

Ermittelt die numerische Id für einen Menüpunkt innerhalb einer Dialogbox.

`dialog`: Bezeichnung der Dialogbox, dessen Menü betrachtet werden soll.

`path`: Pfad über beliebig viele Popups bis zu einem Menüpunkt wird als Einträge in einem `Line`-Objekt beschrieben. (Beispiel: „Datei“, „Öffnen“)

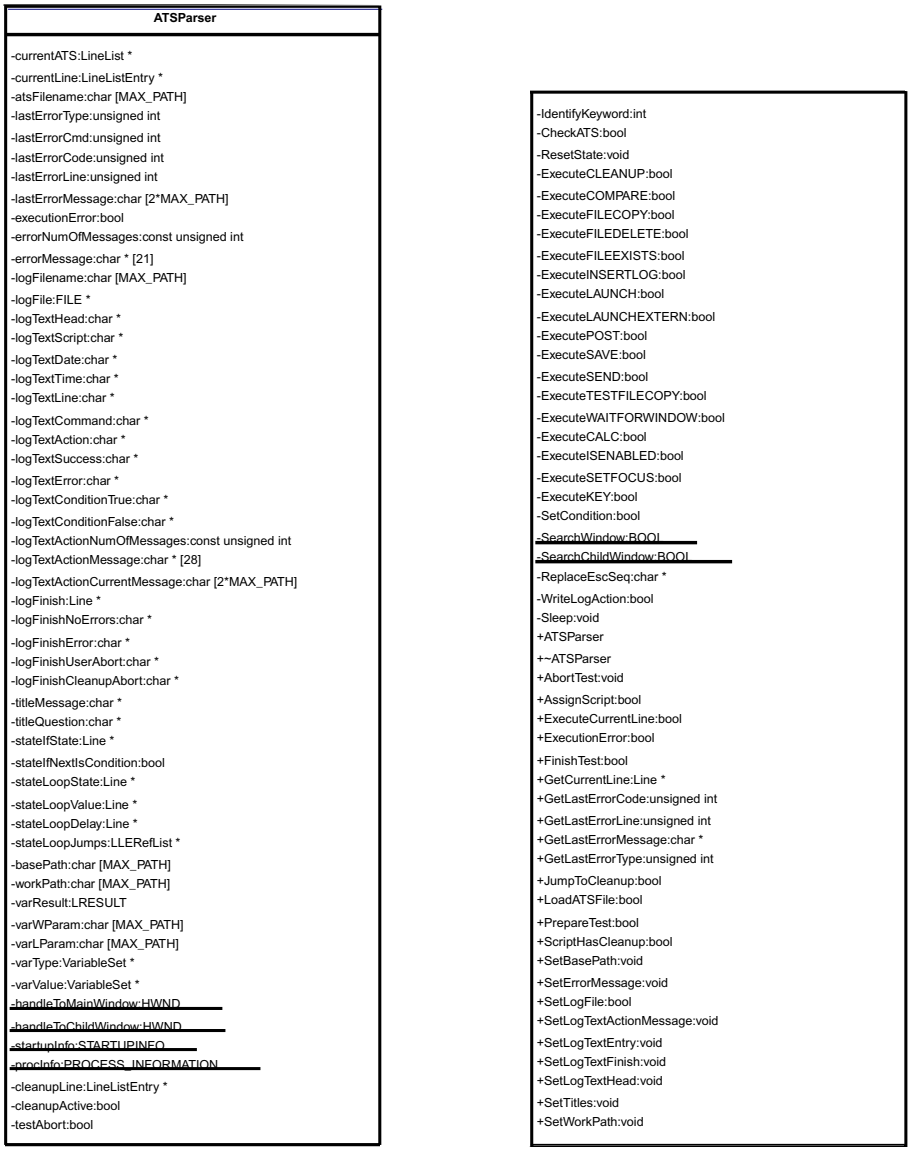
`buffer`: Puffer in dem die numerische Id in Form einer Zeichenkette gespeichert werden soll.

`size`: Größe des Puffers.

`Rückgabewert`: Wenn die Dialogbox ein Menü besitzt und der angegebene Pfad innerhalb des Menüs gültig ist, wird `true` geliefert.

Klasse `ATSParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse bietet die Möglichkeit ATS-Skripte aus Dateien oder `LineList`-Objekten einzuladen und auszuführen. Während der Kommandoausführung können Protokolle angelegt werden.

3. Beschreibung wesentlicher Methoden

`void AbortTest(void)` Der aktuelle Testlauf wird abgebrochen.

`bool AssignScript(LineList& script)` Weist dem Parser ein ATS-Skript in Form eines `LineList`-Objektes zu.

`script:` Referenz auf ein `LineList`-Objekt (ATS-Skript).

`Rückgabewert:` Wenn das Script fehlerfrei geladen wurde und syntaktisch korrekt ist, wird `true` geliefert.

`bool ExecuteCurrentLine(void)` Führt die aktuelle Zeile im ATS-Skript aus.

`Rückgabewert:` Wenn eine die aktuelle Zeile gültig ist und bei der Ausführung kein Fehler auftrat, wird `true` zurückgegeben.

`Line* GetCurrentLine(void)` Ermittelt die aktuelle Skript-Zeile und liefert eine Kopie in Form eines `Line`-Objektes.

`Rückgabewert:` Wenn die aktuelle Zeile gültig ist, wird ein Zeiger auf ein `Line`-Objekt geliefert, das eine Kopie dieser Zeile enthält.

`bool JumpToCleanup(void)` Setzt die aktuelle Zeile auf das `CLEANUP`-Kommando des aktuellen Skriptes, falls vorhanden.

`Rückgabewert:` Wenn es ein `CLEANUP`-Kommando gibt und die Ausführung noch nicht unterhalb von `CLEANUP` ist, wird `true` zurückgegeben.

`bool LoadATSFile(const char* atsFile)` Wenn das Script fehlerfrei geladen wurde und syntaktisch korrekt ist, wird `true` geliefert.

`atsFile:` Name der ATS-Skriptdatei.

`Rückgabewert:` Konnte die Datei erfolgreich geladen werden, wird `true` zurückgegeben.

`bool ScriptHasCleanup(void)` Ermittelt ob das aktuell geladene Skript ein `CLEANUP`-Kommando enthält.

`Rückgabewert:` Wenn ein Skript geladen ist und dieses ein `CLEANUP`-Kommando enthält, wird `true` zurückgegeben.

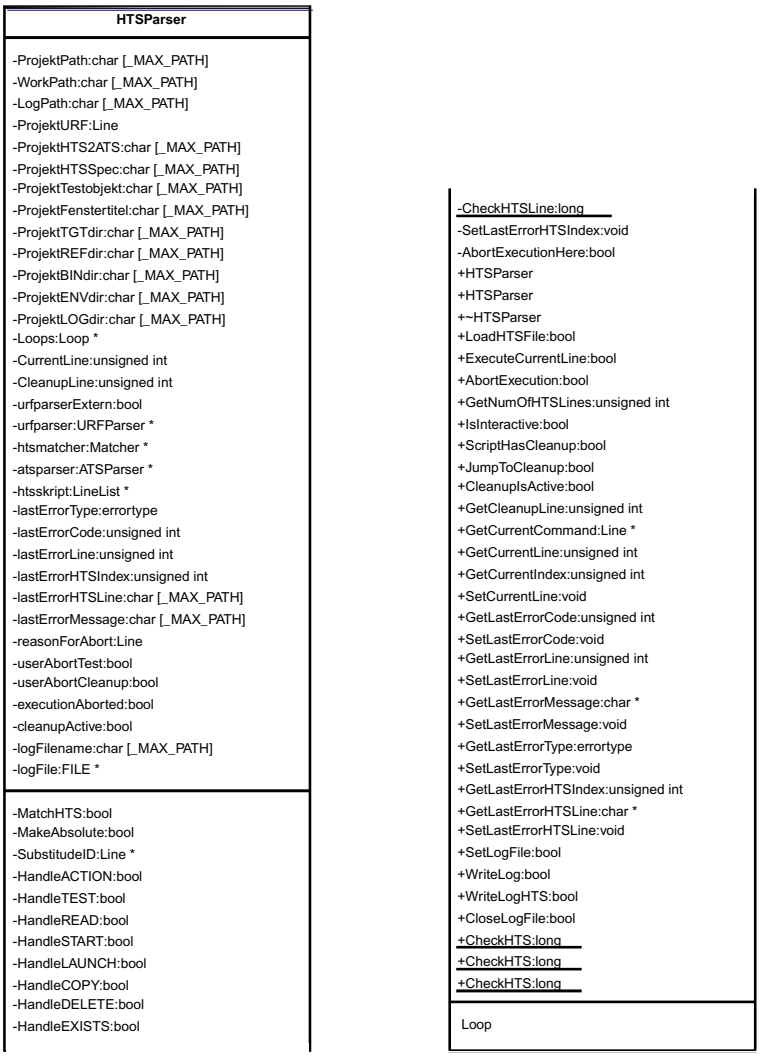
`bool SetLogFile(const char* fname)` (De-)aktiviert die Logfunktion und legt den Namen der Logdatei fest.

`fname:` Name der Log-Datei. Wenn dieser Parameter 0 ist, wird die aktuelle Log-Datei geschlossen und die Log-Funktion deaktiviert.

`Rückgabewert:` Bei erfolgreichem Öffnen bzw. Schließen der Logdatei wird `true` geliefert.

Klasse `HTSParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse bietet die Möglichkeit HTS-Skripte aus Dateien einzuladen und auszuführen. Während der Kommandoausführung können Protokolle angelegt werden.

3. Beschreibung wesentlicher Methoden

`HTSParser(Line URF, const char* RUL, const char* SPC, const char* Wd, const char* Pd, const char* To, const char* Ft)`

Über die Parameter des Konstruktors werden Informationen wie Verzeichnisse und Namen von Regeldateien übergeben, die zur Ausführung von Skriptdateien wesentlich sind.

`URF:` Liste der URF-Dateien (Ressourcen-Datenbanken)

`SPC:` Name der Regeldatei zur Definition der Sprachsyntax (z.B. `hts.syn`)

`RUL:` Name der Regeldatei zur Abbildung HTS→ATS

`Wd:` Name des Arbeitsverzeichnisses, in dem das Testobjekt liegt (absoluter Pfad mit einem Backslash am Ende)

`Pd:` Name des Projektverzeichnisses, in dem die Testskripte, Referenzdateien etc. liegen (absoluter Pfad mit einem Backslash am Ende)

`To:` Name des Testobjektes (z.B. „DEVELOP.EXE“)

`Ft:` Fenstertitel des Testobjektes (z.B. „Steuerprogramm“)

`bool LoadHTSFile(const char* htsfile)` Lädt eine HTS-Datei in den Speicher und ruft den Syntax-Check auf. Ermittelt außerdem die CLEANUP-Zeile und Interaktivitätszustand des Skriptes.

`htsfile:` Name der Skriptdatei (HTS-Datei).

`Rückgabewert:` Wenn das Script fehlerfrei geladen wurde und syntaktisch korrekt ist, wird `true` geliefert.

`bool ExecuteCurrentLine(void)` Führt die aktuelle Zeile im HTS-Skript aus.

`Rückgabewert:` Wenn eine die aktuelle Zeile gültig ist und bei der Ausführung kein Fehler auftrat, wird `true` zurückgegeben

`bool ExecuteCurrentLine(void)` Führt die aktuelle Zeile im HTS-Skript aus.

`Rückgabewert:` Wenn die aktuelle Zeile gültig ist und bei der Ausführung kein Fehler auftrat, wird `true` zurückgegeben

`void AbortExecution(void)` Der aktuelle Testlauf wird abgebrochen.

`bool IsInteractive(const char* htsfile)` Ermittelt ob ein HTS-Skript mindestens einen interaktiven Befehl enthält. (z.B. `MESSAGE`, `QUESTION`)

`htsfile:` Name der Skriptdatei (HTS-Datei).

`Rückgabewert:` Wenn das Skript mind. einen interaktiven Befehl enthält wird `true` zurückgegeben.

`bool ScriptHasCleanup(void)` Ermittelt ob das aktuell geladene Skript ein `CLEANUP`-Kommando enthält.

`Rückgabewert:` Wenn ein Skript geladen ist und dieses ein `CLEANUP`-Kommando enthält, wird `true` zurückgegeben.

`bool JumpToCleanup(void)` Setzt die aktuelle Zeile auf das `CLEANUP`-Kommando des aktuellen Skriptes, falls vorhanden.

`Rückgabewert:` Wenn es ein `CLEANUP`-Kommando gibt und die Ausführung noch nicht unterhalb von `CLEANUP` ist, wird `true` zurückgegeben.

`bool SetLogFile(const char* fname)` (De-)aktiviert die Logfunktion und legt den Namen der Logdatei fest.

`fname:` Name der Log-Datei. Wenn dieser Parameter 0 ist, wird die aktuelle Log-Datei geschlossen und die Log-Funktion deaktiviert.

`Rückgabewert:` Bei erfolgreichem Öffnen bzw. Schließen der Logdatei wird `true` geliefert.

`static long CheckHTS(LineList* hts, const char* syntax, Line urf, const char* mainWindow, LineList* result)`

Überprüfen der Syntax eines vorliegenden HTS-Skriptes mit Hilfe einer Regeldatei und sicherstellen der Konsistenz benutzter Oberflächen-Elemente mit Ressourcen-Datenbanken.

`hts:` HTS-Skript in Form eines `LineList`-Objektes, welches auf gültige Syntax überprüft werden soll.

`syntax:` Dateiname einer Regeldatei zur Syntaxbeschreibung (z.B. `hts.syn`)

`urf:` Liste von URF-Dateien (incl. Pfaden) in Form eines `Line`-Objektes. Die angegebenen Ressourcen-Datenbanken dienen zur Konsistenzprüfung benutzter Bezeichner für Oberflächen-Elemente in den Skriptkommandos .

`mainWindow:` Titel des Hauptfensters (z.B. „Steuerprogramm“)

`result:` Zeiger auf ein `LineList`-Objekt zur Aufnahme von Fehlermeldungen, die beim Überprüfen des HTS-Skriptes aufgetreten sind.

`Rückgabewert:` Das Bitmuster des long-Wertes gibt Informationen über das überprüfte Skript:

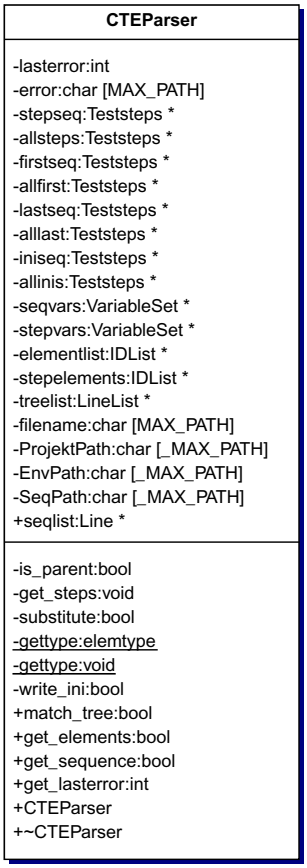
Bit 0 gesetzt: Ein Fehler ist aufgetreten

Bit 1 gesetzt: Das Skript hat ein `CLEANUP`-Kommando

Bit 2 gesetzt: Das Skript enthält interaktive Kommandos

Klasse `CTEParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse bietet Funktionalitäten zur Verarbeitung eines CTE-Diagrammes zur automatisierten und systematischen Gewinnung von Testsequenzen. Das Diagramm wird beim Anlegen eines Objektes dieser Klasse als Baumstruktur in den Speicher eingelesen. Mit dieser Grundlage können Testskripte aus den Attributen der Elemente für beliebige Testfälle bzw. Testsequenzen (Kombinationstabelle des CTE-Diagrammes) generiert werden.

3. Beschreibung wesentlicher Methoden

`CTEParser (const char *f, const char* Pd)` Die Parameter des Konstruktors bestimmen den Dateinamen des CTE-Diagrammes und das Verzeichnis eines *ATOS*-Projektes.

`f:` Name des CTE-Diagrammes (Endung `.cte`).

`Pd:` Gibt den absoluten Pfad zu einem *ATOS*-Projektverzeichnis an. Generierte Skripte und Konfigurationsdateien müssen in Verzeichnisse kopiert werden, die *ATOS* zur Zeit der Testdurchführung bekannt sind. Testskripte werden in \`seq`\, Konfigurationsdateien in \`env`\ unterhalb des in `Pd` angegebenen Pfades angelegt.

`bool match_tree(void)` Filtert das CTE-Diagramm mit Hilfe der Klassen `Matcher` (Regeldatei `cte.rul`) und `LineParser`, um eine leicht handhabbare Baumstruktur zu gewinnen. Diese Baumstruktur wird in einer verketten Liste von Zeilen als Objekt der Klasse `LineList` aufbewahrt.

`Rückgabewert:` Wenn das Klassifikationsdiagramm aus der Datei im Speicher als verkettete Liste erfolgreich angelegt werden konnte, wird `true` zurückgegeben.

`bool get_elements(void)` Die Elemente mit ihren Attributen werden aus der vorliegenden Baumstruktur (`LineList`-Objekt) als verkettete Speicherstruktur abgelegt.

`Rückgabewert:` Wenn alle Elemente als Speicherstrukturen erfolgreich angelegt und verlinkt werden konnten, wird `true` zurückgegeben.

`bool get_sequence(char *testname)` Generiert ein Testskript und ggf. zugehörige Konfigurationsdateien für einen Testfall bzw. für eine Testsequenz mit der Bezeichnung `testname` aus den Attributen der Elemente einer vorliegenden Baumstruktur.

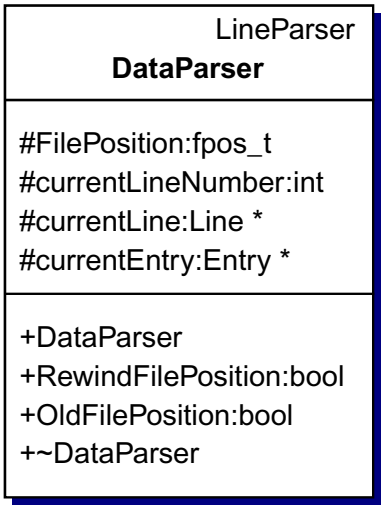
`testname:` Zeiger auf den Bezeichner eines Testfalles bzw. einer Testsequenz aus der Kombinationstabelle des CTE-Diagrammes.

`Rückgabewert:` Wenn Testsequenz und Konfigurationsdateien erfolgreich generiert werden konnten, wird `true` zurückgegeben.

4.2.2 Komponente *DataDiff*

Klasse `DataParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse erweitert Funktionalitäten der Klasse `LineParser` zur Anpassung auf die Bedürfnisse des Dateivergleichers *DataDiff*.

3. Beschreibung wesentlicher Methoden

`bool RewindFilePosition(Line* aktZeile, Entry* aktEintrag)` Setzt die Leseposition der Datei an den Anfang und speichert aktuelle Zeile und Eintrag für `OldFilePosition()`.

`aktZeile:` Zeiger auf ein `Line`-Objekt zur Aufnahme der aktuellen logischen Zeile.

`aktEintrag:` Zeiger auf ein `Entry`-Objekt zur Aufnahme des aktuellen Wortes (`Entry`-Objekt) einer logischen Zeile.

`Rückgabewert:` Wenn die aktuelle Leseposition der Datei ermittelt werden konnte, wird `true` zurückgegeben.

`bool OldFilePosition(Line*& aktZeile, Entry*& aktEintrag)` Setzt die Leseposition der Datei, die logische Zeile und den aktuellen Eintrag auf die Werte, die beim Aufruf von `RewindFilePosition()` ermittelt wurden.

`aktZeile`: Referenz eines Zeigers auf ein `Line`-Objekt mit der aktuellen logischen Zeile.

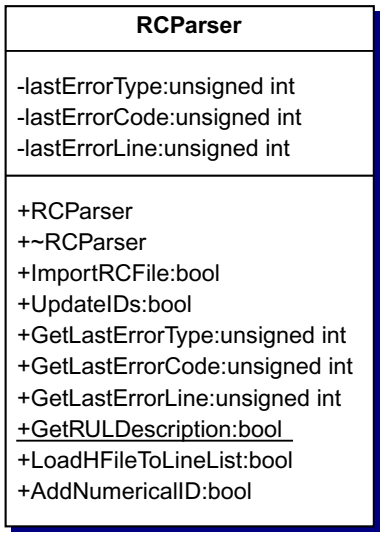
`aktEintrag`: Referenz Zeiger auf ein `Entry`-Objekt mit dem aktuellen Wort (`Entry`-Objekt) der aktuellen logischen Zeile.

`Rückgabewert`: Wenn die aktuelle Leseposition der Datei erfolgreich zurück gesetzt werden konnte, wird `true` zurückgegeben.

4.2.3 Komponente *RCParser*

Klasse `RCParser`

1. UML-Klassendiagramm



2. Verantwortlichkeit der Klasse

Die Klasse bietet die Möglichkeit zur Bildung einer Datenbank (`.urf`-Datei) über Oberflächen-Elemente einer Windowsanwendung, die in ihren RC- und H-Dateien vorgefunden werden konnten.

3. Beschreibung wesentlicher Methoden

`bool ImportRCFile(const char* rulFile, const char* rcFile, LineList* urf)`

Lädt eine RC-Datei mittels Funktionen der Klassen `Matcher` und `LineParser` in ein Objekt der Klasse `LineList`. Jede angelegte logische Zeile matched mit einer Regel aus der Datei `rulFile`. Durch die Konstruktion verschiedener Regeldateien (Importfilter) können abweichende Dialekte der RC-Dateien bei unterschiedlichen Entwicklungsumgebungen (Borland, Microsoft) berücksichtigt werden.

`rulFile:` Regeldatei, die als Importfilter verwendet werden soll.

`rcFile:` RC-Datei, deren Daten untersucht werden sollen.

`urf:` Zeiger auf ein `LineList`-Objekt, in dem die Informationen über vorgefundene Oberflächen-Elemente abgelegt werden sollen.

`Rückgabewert:` Verlief die Generierung der Ressourcen-Datenbank erfolgreich, wird `true` zurückgegeben.

`bool UpdateIDs(const char* hFile, LineList* urf)` Bildet die Zuordnung zwischen Bezeichnern von Oberflächen-Elementen und ihren numerischen Pendants. Dazu werden die `#define`-Einträge aus einer H-Datei mit den eindeutigen ID-Bezeichnern aus der Ressourcen-Datenbank verglichen.

`hFile:` H-Datei, in der die Zuordnung zwischen ID-Bezeichnern und numerischen IDs mittels `#define`-Einträge definiert wird.

`urf:` Zeiger auf ein `LineList`-Objekt, in dem die Informationen über vorgefundene Oberflächen-Elemente gespeichert sind (Ressourcen-Datenbank).

`Rückgabewert:` Wenn keine Fehler beim Öffnen der H-Datei oder beim Bearbeiten der Ressourcen-Datenbank auftraten, wird `true` zurückgegeben.

`bool LoadHFileToLineList(const char* hFile, LineList* list)` Lädt die Zuordnungen zwischen ID-Bezeichnern und numerischen IDs aus einer H-Datei in eine Liste.

`hFile:` H-Datei, in der die Zuordnung zwischen ID-Bezeichnern und numerischen IDs mittels `#define`-Einträge definiert wird.

`list:` Zeiger auf ein `LineList`-Objekt, in dem die Zuordnungen gespeichert werden sollen.

`Rückgabewert:` Wenn keine Fehler beim Öffnen der H-Datei oder beim Speichern in die Liste auftraten, wird `true` zurückgegeben.

`bool AddNumericalID(Line* object, LineList* list)` Fügt einer Zeile aus der Ressourcen-Datenbank eine numerische ID zu, die aus einer Zuordnungsliste ermittelt wurde.

`object:` Eine Zeile aus der Ressourcen-Datenbank, der eine numerische ID aus der Zuordnungsliste zugewiesen werden soll.

`list:` Zeiger auf ein `LineList`-Objekt, in dem die Zuordnungen vorliegen und zuvor mit `LoadHFilesToLineList()` erstellt wurde.

`Rückgabewert:` Wenn keine Fehler beim Auffinden und Zuweisen der numerischen ID auftraten, wird `true` zurückgegeben.

4.3 Umfang der Implementierung

Komponente		**Umfang in LOC**
ATOS	GUI-Komponenten	11000
	Klasse ATOS	7500
	Klasse ATSParser	6000
	Klasse HTSParser	3900
	Klasse CTEParser	1700
RCParser	GUI-Komponenten	5700
	Klasse RCEditor	6300
	Klasse RCParser	600
DataDiff	Hauptprogramm / Hilfsklassen	600
Hilfsklassen		7700
Summe		**51000**

Stand der Implementierung vom 18.11.2002

Kapitel 5

Bewertung

5.1 Vollständigkeit der Testfälle

In diesem Kapitel soll ein Verfahren entwickelt werden, um die minimale Überdeckung aller ausführbaren Funktionalitäten mit Testskripten in Umgebung der Hardwaresimulation zu erreichen. Die entworfenen Testfälle aus Kapitel 3.3.3 entstanden aus den Verhaltensspezifikationen der bekannten Anwendungsfälle. Dabei kann nicht sichergestellt werden, dass alle erreichbaren Programmfunktionen mit den definierten Testfälle auch wirklich abgedeckt wurden. Um diese Testlücken aufzudecken, ist es notwendig, den Quelltext vor der Übersetzung in ein ausführbares Programm so zu modifizieren, dass jede aufgerufene Funktion protokolliert wird. In der Fachsprache ist dieses Verfahren unter der Bezeichnung *Instrumentierung* bekannt und wird in der Praxis vor allem zum Aufspüren von Programmierfehlern (Debugging) eingesetzt.

Auf dem instrumentierten Testling sind alle definierten Testfällen abzufahren, um daraus Aussagen über die Vollständigkeit der Testfälle treffen zu können. Funktionen, die sich als „benutzt“ erwiesen haben und im Rahmen der Umgebungssimulation ausführbar sind, können mit neuen Testfällen abgedeckt werden. Dieser Zyklus ist in Abbildung 5.1 dargestellt.

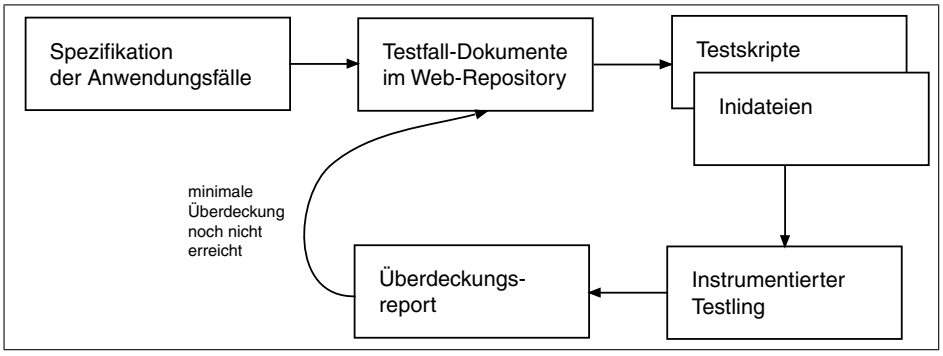


Abbildung 5.1: Maximierung des Überdeckungsgrades

5.1.1 Quelltext - Instrumentierung

Die Instrumentierung des Quellcodes kann in Abhängigkeit des zu erreichenden Zieles sehr unterschiedlich ausfallen. In der Softwaretechnik unterscheidet man zwischen verschiedenen kontrollflussbezogenen Verfahren. Der C0-Test dient der Überprüfung, ob alle Anweisungen eines Programmes ausgeführt wurden. Weil schon einfache, bedingte Anweisungen nicht beachtet werden, ist er nicht sehr aussagekräftig. Die Zweigüberdeckung eines C1-Tests hingegen überprüft, ob alle möglichen Zweige einer Bedingung ausgeführt wurden. Eine Schwachstelle bei diesem Verfahren ist, dass komplexe zusammengesetzte Bedingungen nicht beachtet werden. Die einfache und mehrfache Bedingungsüberdeckung (C2- und C3-Test) versucht durch Kombination der Bedingungen diese Schwachstelle zu umgehen. Für einen erschöpfenden Test bietet sich der C4-Test zur Messung der Pfadüberdeckung an. Hierbei wird untersucht, ob alle möglichen Pfade eines Kontrollflussgraphen durchlaufen wurden.

Alle aufgezählten Verfahren zielen auf die Optimierung des Entwurfs von WhiteBox-Testfällen ab. Voraussetzung dafür ist die Kenntnis der programminternen Struktur, um Kontrollflussgraphen untersuchter Code-Teile zu untersuchen und dabei Fehler ausfindig zu machen.

Mit den entworfenen BlackBox-Testfällen des Regressionstests soll der Aufruf von essentiellen Programmfunktionen sichergestellt werden, weshalb eine C0-, C1-, C2-, C3- oder C4-Instrumentierung in unserem Falle nicht geeignet wäre. Jede Aktion und Reaktion des XCTL-Systems wird durch eine Menge von Funktionen ausgelöst. Diese Funktionen lassen sich ihrerseits bestimmten Anwendungsfällen zuordnen. Umgekehrt lassen sich für einen Anwendungsfall leicht Testfälle konstruieren, die bei ihrer Durchführung noch

nicht abgedeckte Funktionen aufrufen. Da hierfür ein fundiertes Wissen über die Programmquellen benötigt wird, spricht man auch von einem GreyBox-Test. (siehe Kapitel 2.1).

Eine Instrumentierung der Quellen mit dem Ziel, die funktionale Überdeckung zu ermitteln, also den Aufruf der Programmfunktionen zu überwachen, ist demnach die geeignete Methode. Die Suche nach einem entsprechenden Werkzeug stellte sich dabei als sehr schwierig heraus. Die meisten Programme bekannter Hersteller unterstützen nur Quellcode für Microsoft Visual C++. In den wenigsten Fällen wird der Borland-Dialekt unterstützt, dann aber fast ausschließlich für 32-Bit Programme. Die Tabelle 5.1 soll eine kurze Übersicht über das Angebot aktueller Werkzeuge geben.

Werkzeug	**Unterstützung**
C-Cover, Bullseye	kein 16-Bit
DeepCover, Cigital	nicht mehr erhältlich
Visual Testing Toolset, McCabe	keine EvaluationCopy erhältlich
Pure Coverage, Rational Software	nur MSVC++
Visual Test, Rational Software	nur MSVC++
TestWorks (TCAT C/C++), Software Res.	nur MSVC++
Panorama, ISA Inc.	kein Borland
Insure++, ParaSoft	kein Borland (aber cc, gcc, acc)
LDRA Testbed, LDRA	EvaluationCopy nur gegen Bezahlung
CTC++, Testwell Oy	16-Bit, Borland !!

Tabelle 5.1: Werkzeuge zur dynamischen Überdeckungsanalyse

Eine Auflistung aktueller Instrumentierungswerkzeuge und anderer nützlicher Programme für die Bereiche Testen und Debugging ist unter [22] und [23] zu finden.

Die intensive Suche im Internet führte uns auf das Tool *CTC++* der finnischen Firma *Testwell Oy*. Nach Rücksprache mit einem Mitarbeiter erhielten wir eine zeitlich begrenzte Lizenz für Studienzwecke bis zum 31. August 2002. In einer alten MS-DOS Version (v. 4.3) vom 6. Oktober 1997 unterstützt das Werkzeug C und C++ Quellcode für 16-Bit Borland-Projekte. Es bietet die folgenden Instrumentierungsvarianten an: „function coverage“, „decision coverage“, „multicondition coverage“, „interface coverage“ und „timing“.

Mit der Variante „timing“ ist es möglich, die Ausführungszeiten von Funktionen zu bestimmen. Um den Aufruf aller implementierten Klassenmethoden nachzuweisen, hilft die Analyse der „interface coverage“. Eine Instrumentierung zur Untersuchung der „decision coverage“ und der „multicondition coverage“ ist für komplexere .CPP-Dateien des XCTL-Projekts leider nur begingt anwendbar. Der Borland-Compiler kann die vielen globalen Konstanten in den instrumentierten Funktionen nicht verarbeiten. Das Werkzeug ist scheinbar mit Projekten dieser Größe überfordert. C2- bzw. C3-Tests können demnach nur mit Teilmengen des gesamten Quellcodes durchgeführt werden. Die für uns wichtige Instrumentierungsvariante „function coverage“ wird angeboten und ist auch auf das gesamte XCTL-Projekt anwendbar.

Neben *CTC* zum Instrumentieren der Quelldateien liegen die Programme *CTCPOST* zur Bildung von lesbaren ASCII-Reportdateien aus der binären Monitordatei, sowie *CTCHTML* zur Umwandlung der ASCII-Reportdateien in HTML-Dateien bei. Der Umgang mit dem Tool wird in Abbildung 5.2 am Beispiel des XCTL-Projektes schematisiert.

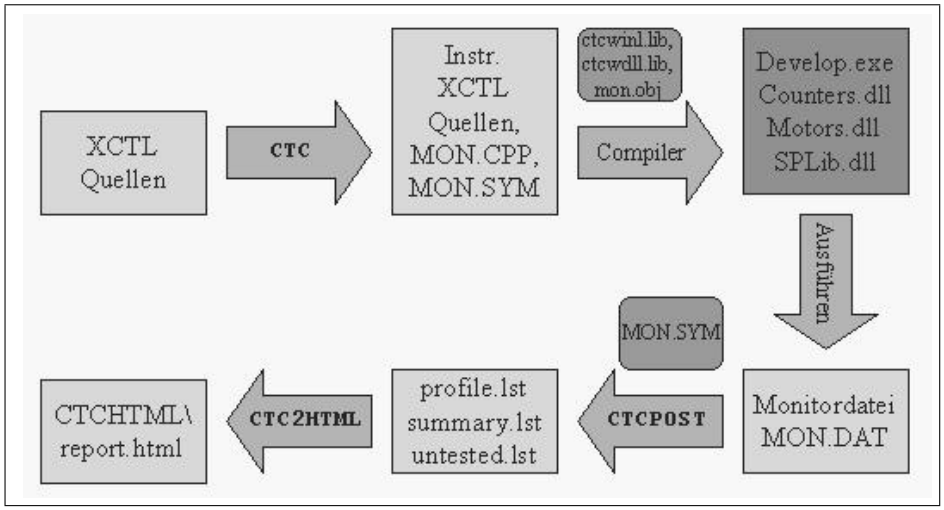


Abbildung 5.2: Arbeitsweise des Instrumentierungswerkzeugs CTC++

Arbeitsweise

Die Quellen des zu instrumentierenden Programmes werden zunächst vom CTC-Precompiler bearbeitet. Hierbei werden je nach gewählter Überdeckungsmethode unterschiedlich viele Messpunkte gesetzt. Zusätzlich ent-

steht eine Symboltabelle `MON.SYM`, in der alle registrierten Messpunkte verwaltet werden. Die Variablendatei `MON.CPP` implementiert die Zähler und wird als Objektfile in jede ausführbare Datei bzw. DLL eingebunden. Protokollierungsfunktionen stellen die mitgelieferten Bibliotheken `ctcwinl.lib` und `ctcwdll.lib` bereit.

Nach Ausführen und Beenden der instrumentierten Programmversion wird die Monitordatei `MON.DAT` ausgeschrieben. Zusammen mit der Symboltabelle `MON.SYM` erstellt das Tool *CTCPOST* die ASCII-Protokolle `profile.lst`, `summary.lst` und `untested.lst`. Das Perlskipt *CTC2HTML* überführt diese Reportdateien schließlich in den Webstandard HTML.

Leider stellte sich die Anwendung des Tools *CTC++* auf unser Projekt, trotz weitestgehender Kompatibilität, als relativ schwierig heraus. Eine grafische Benutzerschnittstelle steht nicht zur Verfügung. Alle Quellen müssen über die Kommandozeile instrumentiert, kompiliert und gelinkt werden. Auch die Auswertung der Überdeckungsergebnisse wird über die MS-DOS-Eingabebox vorgenommen. Zur Automatisierung dieser Arbeitsschritte entwickelten wir eine Regeldatei mit über 2000 Zeilen für das borlandintegrierte Kommandozeilenprogramm *make*. Den Ausgangspunkt dafür bildete ein automatisch generiertes Makefile für das XCTL-Projekt durch die Borland-Entwicklungsumgebung. Die generierte Datei wurde bereinigt und mit Abhängigkeitsregeln für die Instrumentierung erweitert. Tabelle 5.2 faßt die möglichen Kommandos zusammen.

`make xcontrol` bzw. nur `make`	Erstellen einer nicht instrumentierten Programmversion
`make ctc`	Erstellen einer instrumentierten Programmversion
`make lst`	Erstellen einer ASCII-Reportdatei aus der Monitordatei (mit *CTCPOST*)
`make htmprofile`	Erstellen einer HTML-Reportdatei aus der Datei profile.lst (mit *CTCHTML*)
`make htmsummary`	Erstellen einer HTML-Reportdatei aus der Datei summary.lst (mit *CTCHTML*)
`make htmuntested`	Erstellen einer HTML-Reportdatei aus der Datei untested.lst (mit *CTCHTML*)
`make clean`	Löschen angelegter Objekt-, Monitor-, Reportdateien etc.

Tabelle 5.2: CTC++ Makefile für das XCTL-Projekt

Konfiguration

In enger Verbindung zum Instrumentierungsvorgang steht die Konfigurationsdatei `CTC.INI`, welche sogar zur Ausführungszeit des Testlings im erwarteten Verzeichnis zur Verfügung stehen muss. Das bedeutet weiterhin, dass eine instrumentierte Programmversion nach Ablauf der Lizenz nicht mehr ausführbar ist!

Neben der Lizenzierung sind zwei Parameter in der INI-Datei wesentlich. Messpunkte in DLL´s werden nach ihrem Entladen nicht automatisch in die Datei `MON.DAT` ausgeschrieben. Hiefür ist der sogenannte „save-point function name“ auf die Funktion `WEP` (Windows Exit Procedure) zu setzen, welche in jeder DLL definiert ist.

```
EMBED_FUNCTION_NAME=WEP
```

Für eine Zuordnung zwischen instrumentierten Quelldateien und ihren Originalen, sollten die Zieldateien den gleichen Namen tragen. Das entwickelte Makefile erwartet diese im Verzeichnis `ctc.dir`.

```
TARGET_NAME=%BASE%%EXT%
```

Um mit dem entwickelten Makefile in einer beliebigen Umgebung arbeiten zu können, sind noch die Makros `CTCHOME`, `XCTLHOME`, `INCLUDE` und `LIBS` auf die entsprechenden Verzeichnisse zu setzen. Mit `IMODE` wird der Instrumentierungsmodus festgelegt. Das komplette Makefile zur Automatisierung der Arbeitsschritte mit *CTC++* auf das XCTL-Projekt ist im Anhang E.1 zu finden.

Kompatibilität mit den XCTL-Quellen

Die Kompatibilität mit unseren Projektquellen war erfreulich hoch. Nur in zwei .C-Dateien gab es Konflikte zu beseitigen. Da diese Dateien derzeitig überarbeitet und an eine saubere OO-Struktur angepasst werden, ist in zukünftigen Projektversionen nicht mehr mit solchen Schwierigkeiten zu rechnen.

CTC++ unterstützt keine Funktionen, die der „Kernighan & Ritchie 1“ Signatur entsprechen. Dazu soll folgendes Beispiel aus dem Manual von *CTC++* zur Anschauung dienen.

```
Kernighan & Ritchie 1 functions

  CTC++ does not measure coverage of functions defined with K&R 1 style,
  for example following function is not measured:

        int main(argc, argv)                /* K&R 1 */
        int argc;
        char *argv[];
        { .... }

  You should use K&R 2/ANSI function definitions instead, for example:

        int main(int argc, char *argv[])    /* K&R 2/ANSI */
        { .... }
```

Die Funktionen in der Datei `DETECUSE\KISL1.C` sind daraufhin anzupassen.

Assemblerbefehle müssen mit Semikolons abgeschlossen werden, sonst gibt es bei der Instrumentierung der Quellen Schwierigkeiten. Wenn das in der Datei `DETECUSE\KMPT1.C` nachgeholt wird, steht einer erfolgreichen Instrumentierung der Projektquellen nichts mehr im Weg.

Precompiler

CTC++ kann die Quellcode-Dateien in zwei Varianten instrumentieren. Die erste Variante erhält man durch Ersetzung der Preprozessoranweisung durch ein einfaches Copy-Kommando in der Datei `CTC.INI`. Die Instrumentierung erfolgt dann direkt auf den vorgefundenen C/C++ Programmzeilen. Hingegen bearbeitet die zweite Variante den vorcompilierten Quellcode. Obwohl man mit der ersten Variante mitunter schneller zu einer ausführbaren Programmversion kommt, sind seine Nachteile erheblich:

- Bedingte Übersetzung (Conditional Compilation) in Originalquellen kann zu Problemen führen. Beispiel

  ```
  #ifndef __WIN32__
  ...
  #else
  ...
  ```

  Diese Bedingung wird bei Instrumentierung nicht beachtet, was mitunter zur Registrierung von zwei Funktionen mit identischer Signatur führen kann. Normalerweise würde der Precompiler eine Funktion ausblenden.

- Einige Funktionen werden in den Headern (.H-Dateien) definiert und können ohne Precompiler während der Instrumentierung nicht registriert werden, da sie zur Zeit der Instrumentierung nicht sichtbar sind.

- Registrierung der Funktionen erfolgt ohne Klasseninformationen. (Zugehörigkeit der Funktionen zu Klassen durch Qualifizierer, beispielsweise `TDevice::Initialize()`)

- „Interface Coverage“ – Analyse nicht möglich, weil Headerdateien zur Zeit der Instrumentierung nicht gesehen werden.

- Wenn nicht vorübersetzte Quellen illegalen C/C++ Code enthalten, kann die Übersetzung der instrumentierten Quellen fehl schlagen.

Zusätzlich müssen zur Instrumentierung ohne vorherige Bearbeitung durch den Borland-Precompiler alle Zeilenumbruch-Zeichen (\) aus den Quellen entfernt werden, da ansonsten das Instrumentierungswerkzeug diese Zeichen als unbekannt zurückweist.

Viele Gründe sprechen dafür, den Borland-Precompiler vor der Instrumentierung einzusetzen. Im Makefile muss dazu jede Quelldatei mit den selben Compiler-Flags an das Instrumentierungstool *CTC++* übergeben werden, wie bei der Übersetzung von .CPP-Dateien in .OBJ-Dateien. Trotz dieses zusätzlichen Aufwands erhält man nach aufmerksamer Arbeit schließlich eine lauffähige Programmversion, welche alle definierten Messpunkte selbständig protokolliert.

5.1.2 Auswertung der Ergebnisse

In den folgenden Abschnitten sollen die Ergebnisse der funktionalen Überdeckung nach Ablauf aller definierten Testfälle auf einer instrumentierten XCTL-Version vom 20. Januar 2002 diskutiert werden. Dabei sollen Gründe für nicht aufgerufene Funktionen kurz zusammengestellt werden, um eine Aussage über die Qualität der Testfälle treffen zu können.

Mit insgesamt 983 registrierten Funktionen würde eine detailierte Auswertung jeder einzelnen Funktion und Diskussion ihrer Bedeutung innerhalb des Programmes zu weit führen. Im Anhang E.2 findet man eine Übersicht aller registrierten Funktionen und ihren Aufruf-Häufigkeiten nach Durchführung des Regressionstestpaketes.

Im allgemeinen lassen sich nicht abgedeckte Funktionen in eine der folgenden sieben Kategorien einordnen:

1. Tote Funktionen, teilweise unvollständig implementiert, auskommentiert und im Programm nicht benutzt.

2. Methoden, die Funktionalitäten für Objekte einer Klassen bereit stellen, aber nicht benutzt werden.

3. Funktionen, die selbst nur aus toten bzw. unbenutzten Funktionen aufgerufen werden.

4. Virtuelle Funktionen, die in abgeleiteten Klassen überschrieben werden. Innerhalb des Programmes werden nur Objekte der abgeleiteten Klassen instanziiert. Das betrifft zum Beispiel einige virtuelle Motormethoden für die Klasse TC_812, die in den Ableitungen TC_812ISA bzw. TC_812GPIB überschrieben werden. Da kein Objekt der Oberklasse TC_812 während des Programmablaufs angelegt wird, können diese Funktionen auch durch keine Testfälle aufgerufen werden.

5. Funktionen, welche die Anwesenheit realer Hardware erfordern. Das betrifft Detektor- und Motorklassenmethoden, die nicht innerhalb der Umgebungssimulation ausgeführt werden können, sowie alle Programmfunktionalitäten, die auf diese Funktionen bzw. Hardware aufsetzen.

6. Funktionen, die von den bekannten Anwendungsfällen nicht erfasst werden. Dazu zählt zum Beispiel der Justagen-Dialog, mit dem sich Makros ausführen lassen. Für diesen „toten“ Anwendungsfall gibt es keine

Verhaltens-Spezifikationen im Web-Repository und keine bekannte Benutzung im Labor, obwohl die ausführbaren Funktionen in der aktuellen Version immer noch vorhanden sind.

7. Funktionen, die tatsächlich durch keinen Testfall abgedeckt wurden, obwohl sie innerhalb der Umgebungssimulation ausführbar sind und von den Benutzern auch weiterhin benutzt werden.

Die Analyse der Testfallüberdeckung soll die Funktionen des letzten Punktes aufspüren, denn nur hierfür lassen sich neue Testfälle definieren, um den funktionalen Überdeckungsgrad zu maximieren. Alle anderen Funktionen, die mit keinen geeigneten Testfällen mindestens einmal aufgerufen werden können, bilden eine obere Schranke des Überdeckungsgrades.

Abbildung 5.3 zeigt das Ergebnis des Tools CTCHTML, welches auf die Monitordatei (MON.DAT) nach Ablauf des Regressionstestpaketes angewendet wurde. Insgesamt wurden 983 Funktionen in den .C, .CPP und .H Dateien von CTC++ registriert. Nach Ablauf aller Testfälle wurden davon 660 Funktionen mindestens einmal aufgerufen. Das entspricht immerhin 67% aller registrierten Funktionen des gesamten XCTL-Quellcodes.

Instrumentierungslücken

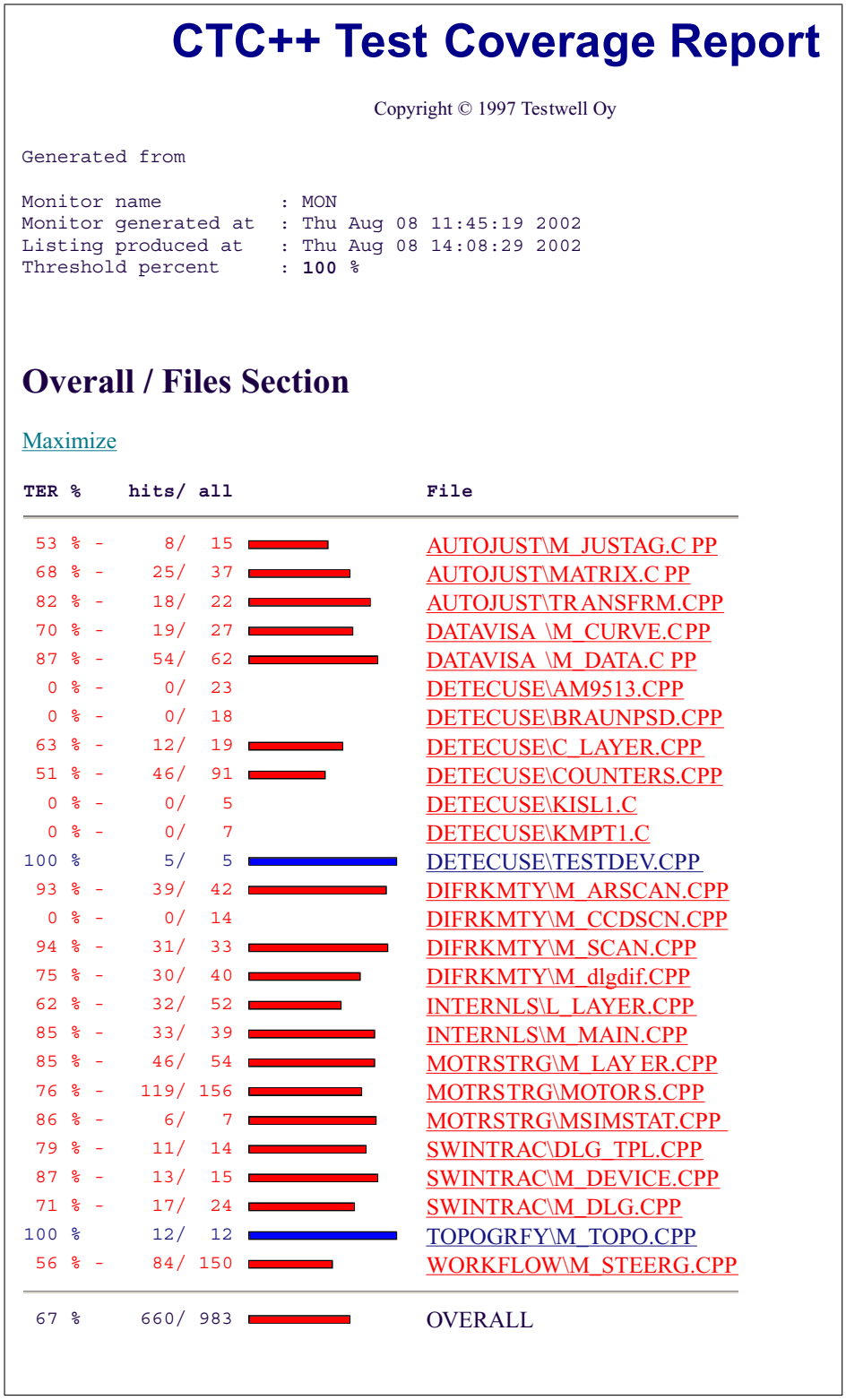
CTC++ kann Funktionen, die in Header-Dateien implementiert wurden in die Instrumentierung einbinden, wenn die Headerdateinamen bis auf die Endung .H mit der jeweiligen .CPP-Datei übereinstimmen. Das ist in unserem Projekt leider nur sehr selten der Fall. Alternativ könnte man das Tool dazu veranlassen, alle includierten Dateien in die Instrumentierung einzubeziehen. Das würde jedoch dazu führen, dass zusätzlich auch alle verwendeten Standard-Bibliotheken in die Messung aufgenommen werden würden.

Weiterhin bietet sich an, die Teile des Quellcodes vor der Instrumentierung auszublenden, in denen Standard-Bibliotheken eingebunden werden. Die Compiler-Direktiven `#pragma CTC SKIP` und `#pragma CTC ENDSKIP` vor und hinter den entsprechenden `#include <...>`'s führen dazu, dass wirklich nur noch Funktionen aus dem Projekt instrumentiert werden. Mit diesem Verfahren werden alle Funktionen des Projektes erfasst. Wie nachfolgend beschrieben, werden Funktionen, die in Headerdateien definiert wurden, für jede .CPP-Datei beim Einbinden dieser Datei erneut registriert. Das Ergebnis waren über 3500 registrierte Funktionen, was verursacht durch die mehrfache Registrierung der Funktionen zu verfälschten Überdeckungsmaßen führt.

Für die Analyse eines realistischen Überdeckungsgrades haben wir uns daher entschieden, nur Funktionen aus den .CPP-Dateien bzw. gleichnamigen .H-Dateien instrumentieren zu lassen. Die meisten definierten Funktionen in den Header-Dateien bestehen aus kurzen Anweisung (*return ...*), um beispielsweise einen Zugriff auf Membervariablen zu ermöglichen. Die wirklich wesentlichen Funktionen des Programmes sind in den .CPP-Dateien definiert und werden von CTC++ registriert und zur Laufzeit überwacht.

Beim Instrumentieren werden wie beschrieben Funktionen, die in (gleichnamigen) Header-Dateien definiert wurden, in jede instrumentierte .CPP-Datei erneut eingebunden und registriert. Diese mehrfache Registrierung der Funktionen lässt sich im Modus der funktionalen Überdeckungsanalyse nicht vermeiden. Die Interface-Überdeckung, bei der jede Klassenmethode genau einmal registriert wird, ist für unsere Analyse jedoch ungeeignet, weil globale Funktionen unbeachtet bleiben würden. Bei der funktionalen Überdeckungsanalyse von Quellcode mit definierten Funktionen in gleichnamigen Headerdateien wird somit die Gesamtzahl der registrierten Funktionen erhöht und der Überdeckungsgrad nach unten verfälscht.

In unserem Projekt betrifft das sieben Methoden, die in der Datei `.\INCLUDE\AUTOJUST\MATRIX.H` definiert und sowohl in `AUTOJUST\M_JUSTAG.CPP`, `AUTOJUST\MATRIX.CPP`, als auch in `AUTOJUST\TRANSFRM.CPP` eingebunden werden. Zwei weitere Methoden in `.\INCLUDE\DETECUSE\TRANSFRM.H` werden sowohl in `DETECUSE\M_JUSTAG.CPP`, als auch in `DETECUSE\TRANSFRM.CPP` eingebunden. Ausserdem werden zwei Methoden aus `.\INCLUDE\DETECUSE\BRAUNPSD.H` sowohl in `DETECUSE\BRAUNPSD.CPP`, als auch in `DETECUSE\COUNTERS.CPP` eingebunden. Zieht man diese 18 zuviel registrierten Funktionen von der Gesamtzahl 983 ab, verbessert sich der Überdeckungsgrad von 67,14% auf 68,39% in unserem Fall nicht wesentlich.



CTC++ Test Coverage Report

Generated from

Monitor name : MON
Monitor generated at : Thu Aug 08 11:45:19 2002
Listing produced at : Thu Aug 08 14:08:29 2002
Threshold percent : **100** %

Overall / Files Section

Maximize

TER %	hits/ all	File
53 % -	8/ 15	AUTOJUST\M_JUSTAG.C PP
68 % -	25/ 37	AUTOJUST\MATRIX.C PP
82 % -	18/ 22	AUTOJUST\TR ANSFRM.CPP
70 % -	19/ 27	DATAVISA \M_CURVE.C PP
87 % -	54/ 62	DATAVISA \M_DATA.C PP
0 % -	0/ 23	DETECUSE\AM9513.CPP
0 % -	0/ 18	DETECUSE\BRAUNPSD.CPP
63 % -	12/ 19	DETECUSE\C_LAYER.CPP
51 % -	46/ 91	DETECUSE\COUNTERS.CPP
0 % -	0/ 5	DETECUSE\KISL1.C
0 % -	0/ 7	DETECUSE\KMPT1.C
100 %	5/ 5	DETECUSE\TESTDEV.CPP
93 % -	39/ 42	DIFRKMTY\M_ARSCAN.CPP
0 % -	0/ 14	DIFRKMTY\M_CCDSCN.CPP
94 % -	31/ 33	DIFRKMTY\M_SCAN.CPP
75 % -	30/ 40	DIFRKMTY\M_dlgdif.CPP
62 % -	32/ 52	INTERNLS\L_LAYER.CPP
85 % -	33/ 39	INTERNLS\M_MAIN.CPP
85 % -	46/ 54	MOTRSTRG\M_LAY ER.CPP
76 % -	119/ 156	MOTRSTRG\MOTORS.CPP
86 % -	6/ 7	MOTRSTRG\MSIMSTAT.CPP
79 % -	11/ 14	SWINTRAC\DLG_TPL.CPP
87 % -	13/ 15	SWINTRAC\M_DEVICE.CPP
71 % -	17/ 24	SWINTRAC\M_DLG.CPP
100 %	12/ 12	TOPOGRFY\M_TOPO.CPP
56 % -	84/ 150	WORKFLOW\M_STEERG.CPP
67 %	660/ 983	OVERALL

Abbildung 5.3: CTC++ Überdeckungsreport - Überblick

Überdeckungslücken

Destruktor-Methoden von Klassen werden nur dann als „aufgerufen“ gewertet, wenn ein Objekt dynamisch mittels `new` erzeugt und mittels `delete` wieder gelöscht wird oder wenn funktionslokale Objekte beim Verlassen einer Funktion zur Laufzeit des Programmes aus dem Speicher entfernt werden. Die Aufrufe von Destruktor-Methoden globaler Objekte werden nicht gezählt, weil sie erst nach Beendigung des Programmes aufgerufen werden, also ausserhalb der Überwachungsmöglichkeiten von CTC++. Aus diesem Grund wird zum Beispiel der Destruktor des globalen Objekts vom Typ TMain beim Programmende scheinbar nicht aufgerufen, was wiederum den berechneten Überdeckungsgrad nach unten verfälscht.

Wie in Abbildung 5.3 zu erkennen, wurden keine Funktionen aus den Dateien `DETECUSE\AM9513.CPP`, `DETECUSE\BRAUNPSD.CPP`, `DETECUSE\KISL1.C`, `DETECUSE\KMPT1.C` und `DIFRKMTY\M_CCDSCN.CPP` aufgerufen. Diese 67 Funktionen sind aus dem oben genannten zweiten Grund mit den definierten Testfällen nicht erreichbar und wurden nur der Vollständigkeit halber in die Instrumentierung aufgenommen. `KISL1.C`, `KMPT1.C` und `BRAUNPSD.CPP` stellen Funktionalitäten für den 0-dimensionalen Detektoren Radicon SCSCS und für den 1-dimensionalen Detektor Braunpsd bereit, die im Rahmen der Umgebungssimulation nicht eingesetzt werden können. Die Datei `AM9513.CPP` stellt Methoden zum Zugriff auf die Zählereinheit Am9513A zur Verfügung. Diese 16-Bit Zähler werden zum Beispiel bei Messungen mit dem russischen 0-dimensionalen Detektor SCSCS (TGenericDevice) benutzt und befinden sich daher ebenfalls außerhalb der Simulations-Möglichkeiten.

Die Methoden zur Funktionalität eines CCD-AreaScans in der Datei `M_CCDSCN.CPP` setzen die Existens eines CCD-Detektors voraus. Die Einbindung solcher 2-dimensionaler Detektoren in das XCTL-System ist noch nicht abgeschlossen, weshalb noch keine sinnvollen Testfälle für diesen Anwendungsfall entwickelt werden konnten.

Die folgenden Tabellen geben Erklärungen bzw. Ursachen für alle nicht abgedeckten Funktionen. Jede Funktion lässt sich in eine der sieben aufgeführten Kategorien einordnen.

AUTOJUST\M_JUSTAG.CPP

Unaufgerufene Funktion	Ursache
TMatrizenListe::ist_leer()	Nicht benutzt.

AUTOJUST\MATRIX.CPP

Unaufgerufene Funktion	Ursache
TMatrix::operator_+()	Nicht benutzt.
TMatrix::operator_-()	Nicht benutzt.
TMatrix::operator_*()	Nicht benutzt.
operator_*()	Nicht benutzt.
TMatrix::invers()	Nicht benutzt.
TMatrix::pop()	Nicht benutzt.

AUTOJUST\TRANSFRM.CPP

Alle Funktionen werden mit den definierten Testfällen überdeckt.

DATAVISA\M_CURVE.CPP

Unaufgerufene Funktion	Ursache
TDataBase::∼TDataBase()	Kein Destruktoraufruf für Objekt dieser Klasse. (Fehler !!)
TCurve::BackStep()	Nicht abgedeckt.
TCurve::ValueAdd()	Nicht benutzt.
TCurve::Save(LPCSTR)	Nicht benutzt.
TCurve::Save(LPCSTR,int,int)	Nur aufgerufen von unbenutzter Funktion TPsdRemoteSync::TPsdRemoteSync().
TCurve::DeleteUnderGround()	Nur aufgerufen von unbenutzter Funktion TCurve::GetPeakProperties().
TCurve::DeleteFlanks()	Nicht benutzt.
TCurve::GetPeakProperties()	Nicht benutzt.

DATAVISA\M_DATA.CPP

Unaufgerufene Funktion	Ursache
TBitmapSource::New()	Nicht benutzt.
TBitmapSource::ProcessBitmapFile()	Nicht benutzt. (DIB-Datei lesen stürzt ab)
TBitmapSource::RenderDIB()	Nur aufgerufen von nicht abgedeckter Funktion TPlotData::RenderFormat().
TPlotData::SetMeasurementArea()	Kein Aufruf der virtuellen Methode. Nur Aufruf der überladenen Methode TAreaScan::SetMeasurementArea().
TPlotData::RenderFormat()	Nur Verwendung bei Zwischenablage. (Aufruf über „Bearbeiten“ → „Kopieren“) Nicht abgedeckt.
TPlotData::RenderAllFormat()	"
TPlotData::DoCopy()	"
TCurveShowParam::Dlg_OnHScrollBar()	„Einstellung für die Darstellung“, BildpunkteX/Y (Scrollbar) nicht abgedeckt.

DETECUSE\C_LAYER.CPP

Unaufgerufene Funktion	Ursache
dlGetVersion()	Nur aufgerufen aus toter Funktion TMain::ShowProgramStatus().
dlGetInstance()	"
dlSetDevice()	Nur aufgerufen aus Funktion TChooseDeviceCmd::TChooseDeviceCmd(). Nicht abgedeckt.
dMeasureStart()	"
dMeasureStop()	"
dSetExposureValues()	"
dGetExposureValues()	"

DETECUSE\TESTDEV.CPP

Alle Funktionen werden mit den definierten Testfällen überdeckt.

DETECUSE\COUNTERS.CPP

Unaufgerufene Funktion	Ursache
TGenericDevice::TGenericDevice()	Nicht ausführbar in Umgebungssimulation !
TGenericDevice::Initialize()	"
TGenericDevice::SetParameters()	"
TGenericDevice::MeasureStart()	"
TGenericDevice::MeasureStop()	"
TGenericDevice::PollDevice()	"
TGenericDevice::InitializeEvent()	"
EventHandler()	"
	(Methode von TGenericDevice)
TGenericDevice::SetSound()	"
TEncoder::TEncoder()	Encoder nicht eingesetzt in Testfällen.
TEncoder::∼TEncoder()	(`Type=Encoder` im INI-File)
TEncoder::Initialize()	Dieser Detektortyp, der Antriebspositionen
TEncoder::GetData()	als Messwerte zurück gibt, soll aus zukünftigen
TEncoder::PollDevice()	Versionen entfernt werden.
TPsd::SetSpezificParametersDlg()	Für Testzähler ist diese Funktion leer.
	Kein Testfall nötig.
TPsd::MeasureStopExternal()	Nicht benutzt.
TPsd::SetAngleRange()	Nicht benutzt.
TStoe_Psd::TStoe_Psd()	Nicht ausführbar in Umgebungssimulation.
TStoe_Psd::∼TStoe_Psd()	"
TStoe_Psd::Initialize()	"
TStoe_Psd::PollDevice()	"
TStoe_Psd::PsdReadOut()	"
TStoe_Psd::PsdInit()	"
TStoe_Psd::PsdStart()	"
TStoe_Psd::PsdStop()	"
TStoe_Psd::PsdRead()	"
TRadicon::TRadicon()	Nicht ausführbar in Umgebungssimulation.
TRadicon::∼TRadicon()	"
TRadicon::Initialize()	"
TRadicon::SetSpezificParametersDlg()	"
TRadicon::MeasureStart()	"
TRadicon::MeasureStop()	"
TRadicon::PollDevice()	"
TRadicon::SetParameters()	"
EventHandler()	"
	(Methode von TRadicon)
TRadicon::InitializeEvent()	"
TRadicon::SetSound()	"
TRadicon::FailureOccured()	"
TScsParameters::TScsParameters()	Nur aufgerufen in nicht ausführbarer Funktion
	TRadicon::SetSpezificParametersDlg()
TScsParameters::Dlg_OnInit()	"
TScsParameters::Dlg_OnCommand()	"
TScsParameters::CanClose()	"
TScsParameters::LeaveDialog()	"

DIFRKMTY\M_ARSCAN.CPP

Unaufgerufene Funktion	Ursache
TAreaScan::rButtonDown()	Funktion zur Darstellung des Kontextmenüs wird durch rechte Maustaste aufgerufen. Nicht abgedeckt.
TAreaScan::ExternSychronized()	Tote Funktion, nur in Ansetzen implementiert und nicht benutzt.
TAreaScan::GetShift()	Aufruf nur wenn eSaveFormat=ShiftedStandard. Dieses Format wird nirgends im Programm gesetzt bzw. benutzt.

DIFRKMTY\M_SCAN.CPP

Unaufgerufene Funktion	Ursache
TScan::rButtonDown()	Funktion zur Darstellung des Kontextmenüs wird durch rechte Maustaste aufgerufen. Nicht abgedeckt.
TScan::SaveDataBase()	Tote Funktion, Rückkehrcode ist immer TRUE und nicht benutzt.

DIFRKMTY\M_dlgdif.CPP

Unaufgerufene Funktion	Ursache
TPsdRemoteSync::TPsdRemoteSync()	Aufruf nur in TAreaScan::ExternSychronized(). Diese Funktion wird in der aktuellen Version nicht benutzt.
TPsdRemoteSync::Dlg_OnInit()	"
TPsdRemoteSync::Dlg_OnTimer()	"
TPsdRemoteSync::PrintFileName()	"
TPsdRemoteSync::GetFileName()	"
TPsdRemoteSync::Dlg_OnCommand()	"
TPsdRemoteSync::CanClose()	"
TPsdRemoteSync::LeaveDialog()	"
TCalibratePsd::Dlg_OnHScrollbar()	„PSD Kalibrierung“ Arbeit mit horizontaler Scrollbar nicht abgedeckt.
TChooseScan::Dlg_OnLButtonUp()	Dialogbox „Scan auswählen“ aktualisieren während aktivem AreaScan nicht abgedeckt.

INTERNLS\L_LAYER.CPP

Unaufgerufene Funktion	Ursache
TAbout::LeaveDialog()	Nur Verwendung im Dialog „Über Steuerprogramm“. (Aufruf über „Hilfe“ → „Über“) Nicht abgedeckt.
TAbout::TAbout()	"
TAbout::Dlg_OnInit()	"
TAbout::Dlg_OnCommand()	"
AboutTheMaker()	"
spGetVersion()	Nur aufgerufen von toter Funktion TMain::ShowProgramStatus().
spGetInstance()	"
GetDirectory()	Nicht benutzt.
GetName()	Nicht benutzt.
SetStatus()	Benutzt in TGenericDevice::PollDevice(), TBraunPsd::PsdReadOut(), in TSteering auskommentiert, in mGetDistance(), wenn bShowEncoder=TRUE. Nicht abgedeckt.
GetBufferLine()	Nicht benutzt.
UnitStr()	Nicht benutzt.
maxi()	Nicht benutzt.
mini()	Nicht benutzt.
maxl()	Nicht benutzt.
minl()	Nicht benutzt.
maxf()	Nicht benutzt.
minf()	Nicht benutzt.
maxd()	Nicht benutzt.
mind()	Nicht benutzt.

INTERNLS\M_MAIN.CPP

Unaufgerufene Funktion	Ursache
TMain::GetVersion()	Nicht benutzt.
ShowProgramStatus()	Nicht benutzt. (Tote Funktion)
MenuSelect()	Wird durch automatisierte Testfälle nicht abgedeckt, weil Menüfunktionen direkt und nicht über Popups aufgerufen werden.
TMain::∼TMain()	Nur ein globales Objekt mit diesem Typ.
TMain::TellMessage()	Nicht benutzt.
TMain::GetDataDlg()	Nicht benutzt.

MOTRSTRG\MOTORS.CPP

Unaufgerufene Funktion	Ursache
TMList::∼TMList()	Kein Destruktoraufruf für Objekt. (Fehler !!)
TPosControl::HScrollbar()	„Verfahren nach Encoder-Position“, Arbeit mit horizontaler Scrollbar nicht abgedeckt. (Funktioniert nicht !!)
TMotor::funcd()	Nur benutzt in TMotor::rtsave()
TMotor::rtsave()	Nur benutzt in TMotor::Translate(long& ,double), wenn eCorrStatus=CorrPolynom, TMotor::Translate(long& ,double) Nur aufgerufen in TMotor::MoveToAngle(). TMotor::MoveToAngle() aufgerufen in TCalibrate::Dlg_OnCommand(), TMotor::PopSettings() und mMoveToDistance(). Dort Bedingung niemals eCorrStatus=CorrPolynom. (Fehler: Korrekturpolynom wird niemals einberechnet !!)
TMotor::MoveByPosition()	Nur aufgerufen in TCalibrate::Dlg_OnCommand() für Motor vom Typ TMotor. Kalibrierung der Testmotoren nicht möglich, weshalb diese virtuelle Funktion niemals direkt aufgerufen wird.
TMotor::MoveToPosition()	Nur aufgerufen in TPosControl::Dlg_OnCommand() und TMotor::MoveToAngle() für Motor vom Typ TMotor. Nicht abgedeckt.
TMotor::MoveByAngle()	Nur aufgerufen in unbenutzter Funktion mMoveByDistance()
TDC_Drive::PushSettings()	Nicht benutzt. (TMotor::PushSettings() ist nicht virtuell)
TDC_Drive::PopSettings()	Nicht benutzt. (TMotor::PopSettings() ist nicht virtuell)
TDC_Drive::GetFailure()	Nur aufgerufen in TC_812::IsMoveFinish(), wenn bInquireStatus=FALSE. Dieser Parameter ist nur für 812GPIB-Antriebe definiert. Daher nicht ausführbar in Umgebungssimulation.
TC_812::SetMoment()	Nicht benutzt.
TC_812::GetMoment()	Nicht benutzt.
TC_812::IsMoveFinish()	Kein Aufruf der virtuellen Methode. Es gibt keinen Motor vom Typ TC_812, nur vom Typ TC_812ISA bzw. TC_812GPIB.
TC_812::SetDynamicGain()	Nicht benutzt.
TC_812::GetDynamicGain()	Nicht benutzt.
TC_812::SetTorque()	Nur aufgerufen in unbenutzter Funktion TC_812::SetMoment().
TC_812::GetTorque()	Nur aufgerufen in unbenutzter Funktion TC_812::GetMoment().
TC_812::GetAcceleration()	Nur aufgerufen in mlGetValue() und mGetValue(), wenn vtype=Acceleration. Nirgendwo Aufruf mit vtype=Acceleration, weshalb diese Funktion unbenutzt ist.
TC_812::_GetPosition()	Kein Aufruf der virtuellen Methode. Es gibt keinen Motor vom Typ TC_812, nur vom Typ TC_812ISA bzw. TC_812GPIB.
TC_812::_GetFailure()	..... „ .....

Unaufgerufene Funktion	**Ursache**
TC_812GPIB::TC_812GPIB()	Nicht ausführbar in Umgebungssimulation.
TC_812GPIB::Initialize()	"
TC_812GPIB::CheckBoardOk()	"
TC_812GPIB::Initialize()	"
TC_812GPIB::ExecuteCmd()	"
TC_832::SetHome()	Nicht abgedeckt.
TC_832::StartLimitWatch()	Nicht abgedeckt.
TC_832::StopLimitWatch()	Nicht abgedeckt.
LimitWatch()	Nicht abgedeckt.
TC_832::IsLimitHit()	Nur aufgerufen in nicht ausgeführten Funktionen TC_832::IsIndexArrived()
TC_832::IsIndexArrived()	Nur aufgerufen in TCalibrate::Dlg_OnTimer() für Motor vom Typ TC_832. Nicht abgedeckt.
TC_832::StartToIndex()	Nur aufgerufen in TCalibrate::Dlg_OnCommand() für Motor vom Typ TC_832. Nicht abgedeckt.
TC_832::GetAcceleration()	Nur aufgerufen in mlGetValue() und mGetValue(), wenn vtype=Acceleration Nirgendwo Aufruf mit vtype=Acceleration, weshalb diese Funktion unbenutzt ist.
TC_832::SetLimit()	Nur aufgerufen in TMotorParam::CanClose(). Setzt dwRemoveLimit des Antriebs vom Typ TC_832. Editbox „Limit“ ist immer disabled, damit ist Funktion unbenutzt.
TC_832::GetStatus()	Nicht benutzt.
TC_832::_GetFailure()	Nur aufgerufen in nicht ausführbarer Funktion TDC_Drive::GetFailure().
TC_832::ExecuteCmd()	Nicht benutzt.

MOTRSTRG\M_LAYER.CPP

Unaufgerufene Funktion	Ursache
mlGetVersion()	Nur aufgerufen von toter Funktion TMain::ShowProgramStatus().
mlGetInstance()	"
TC_812GPIB::StartCheckScan()	Nicht ausführbar in Umgebungssimulation.
mlIsServerOK()	Nicht benutzt.
mSetLine()	Nicht benutzt.
mIsRangeHit()	Nur aufgerufen in TGotoPeakCmd::ControlStep(), wenn `fIntensity[0]<Steering.GetNoiseLevel()`. `fNoiseLevel` wird nur im Konstruktor von `TSteering` auf 300 gesetzt bzw. im Dialog „Justage-Einstellungen“ unter „Rauschen“. Diese Einstellung hat dann aber nur Wirkung innerhalb der „Justagen“, die von den Physikern nicht mehr benutzt werden. Im Dialog „Makro ausführen“ ist `fNoiseLevel` immer 300 und kann von keiner Testdetektor-Intensität unterschritten werden. Damit ist diese Funktion im Rahmen der Umgebungssimulation nicht ausführbar.
mMoveByDistance()	Nicht benutzt.
mExecuteCmd()	Nur aufgerufen von toter Funktion TExecuteCmd::Dlg_OnCommand().

MOTRSTRG\MSIMSTAT.CPP

Unaufgerufene Funktion	Ursache
UnIntializeMotorsSimulation()	Wird im Destruktor TMain::∼TMain aufgerufen. Liegt ausserhalb der Überwachungsmöglichkeiten von CTC++.

SWINTRAC\DLG_TPL.CPP

Unaufgerufene Funktion	Ursache
TModalDlg::CanClose()	Kein direkter Aufruf der virtuellen Methode. Alle Dialogboxen haben eine überladene Methode.
TModelessDlg::Dlg_OnInit()	"
TModelessDlg::CanClose()	"

SWINTRAC\M_DEVICE.CPP

Unaufgerufene Funktion	Ursache
TCounterWindow::rButtonDown()	Funktion zur Darstellung des Kontextmenüs wird durch rechte Maustaste aufgerufen. Nicht abgedeckt.
TCounterWindow::lButtonDown()	Leere Funktion wird nicht benutzt.

SWINTRAC\M_DLG.CPP

Unaufgerufene Funktion	Ursache
TGetData::TGetData()	Aufruf nur in toter Funktion TMain::GetDataDlg().
TGetData::Dlg_OnInit()	"
TGetData::CanClose()	"
TExecuteCmd::TExecuteCmd()	Tote Funktion, nicht aufgerufen bzw. benutzt.
TExecuteCmd::Dlg_OnInit()	"
TExecuteCmd::Dlg_OnCommand()	"
TExecuteCmd::LeaveDialog()	"

TOPOGRFY\M_TOPO.CPP

Alle Funktionen werden mit den definierten Testfällen überdeckt.

WORKFLOW\M_STEERG.CPP

Unaufgerufene Funktion	Ursache
TCmd::FirstStep()	Kein direkter Aufruf der virtuellen Methode. Alle Kommandos haben eine überladene Methode.
TCmd::GetShowData()	"
TCmd::ControlStep()	"
TCmd::ReadyStep()	"
TCmd::Ready()	"
TEditWindow::IsModified()	Tote Funktion.
TEditWindow::ClearModify()	Tote Funktion.
TEditWindow::GetCursor()	Tote Funktion.
TEditWindow::ClearWindow()	Tote Funktion.
TEditWindow::DoSearch()	Tote Funktion.
TEditWindow::ClassName()	Tote Funktion.
TEditWindow::GetCharacteristic()	Tote Funktion.
TCmd::DeviceRequest()	Nicht benutzt.
TCmd::GetFailure()	Nicht benutzt.
TInquireCmd::TInquireCmd()	Nicht benutzt. (totes Makro-Kommando)
TInquireCmd::GetShowData()	Nicht benutzt. (totes Makro-Kommando)
TSaveDataCmd::TSaveDataCmd()	Nicht abgedeckt.
TSaveDataCmd::GetShowData()	Nicht abgedeckt.
TSetFileNameCmd::TSetFileNameCmd()	Nicht abgedeckt.
TSetFileNameCmd::GetShowData()	Nicht abgedeckt.
TChooseDeviceCmd::TChooseDeviceCmd()	Nicht abgedeckt.
TChooseDeviceCmd::GetShowData()	Nicht abgedeckt.
TSetupScanCmd::TSetupScanCmd()	Nicht abgedeckt.
TSetupScanCmd::GetShowData()	Nicht abgedeckt.
TControlFlankCmd::∼TControlFlankCmd()	Makro wird direkt aus TTopographyExecute::Dlg_OnCommand() aufgerufen. Dabei wird Kommando-Objekt dynamisch angelegt in TSteering::StartCmdExecution(TCmdTag), aber nach Ausführung nicht wieder gelöscht. (Fehler !!)
TControlFlankCmd::ReadyStep()	Nicht abgedeckt, weil Kommando nicht über Makrodatei gestartet wurde.
TAreaScanCmd::GetShowData()	Nicht abgedeckt, weil Kommando nicht über Makrodatei gestartet wurde.
TSteering::CallMacro()	Tote Funktion.
TSteering::IsMacroCalled()	Tote Funktion.
TSteering::ReturnMacroCall()	Tote Funktion.
TMacroExecute::CanClose()	Nicht benutzt.
TEditWindow::SetFocus()	Tote Funktion.
TEditWindow::TEditWindow()	Tote Funktion.
TEditWindow::∼TEditWindow()	Tote Funktion.
TEditWindow::Create()	Tote Funktion.
TEditWindow::Size()	Tote Funktion.
TEditWindow::SetTitle()	Tote Funktion.

Unaufgerufene Funktion	**Ursache**
TEditWindow::Paint()	Tote Funktion.
TEditWindow::CanClear()	Tote Funktion.
TEditWindow::ReadFile()	Tote Funktion.
TEditWindow::NewFile()	Tote Funktion.
TEditWindow::SaveFile()	Tote Funktion.
TEditWindow::rButtonDown()	Tote Funktion.
TSetAdjustmentParam::TSetAdjustmentParam()	Dialog-Funktionen für „Justage-Einstellungen“ ist zwar aufrufbar, wird aber nicht mehr verwendet. Daher wurden keine Testfälle definiert.
TSetAdjustmentParam::Dlg_OnInit()	„
TSetAdjustmentParam::Dlg_OnCommand()	„
TSetAdjustmentParam::GetParameter()	„
TSetAdjustmentParam::SetParameter()	„
TSetAdjustmentParam::CanClose()	„
TSetAdjustmentParam::LeaveDialog()	„
TAdjustmentExecute::TAdjustmentExecute()	Dialog-Funktionen für „Justagen“ ist zwar aufrufbar, wird aber nicht mehr verwendet. Daher wurden keine Testfälle definiert.
TAdjustmentExecute::Dlg_OnInit()	„
TAdjustmentExecute::Dlg_OnCommand()	„
TAdjustmentExecute::GetNextTask()	„
TAdjustmentExecute::GetParameter()	„
TAdjustmentExecute::LeaveDialog()	„
TAdjustmentExecute::CanClose()	„
TAdjustmentWindow::TAdjustmentWindow()	Fenster-Funktionen für „Justage-Fenster“ ist zwar aufrufbar, wird aber nicht mehr verwendet. Daher wurden keine Testfälle definiert.
TAdjustmentWindow::∼TAdjustmentWindow()	„
TAdjustmentWindow::CanClose()	„
TAdjustmentWindow::Create()	„
TAdjustmentWindow::SetTitle()	„
TAdjustmentWindow::InitializeTask()	„
TAdjustmentWindow::CounterSetRequest()	„
TAdjustmentWindow::SteeringReady()	„
TAdjustmentWindow::rButtonDown()	„

Auswertung

Wie sich bei der Analyse der Reportdateien von CTC++ heraus gestellt hat, sind die meisten nicht abgedeckten Funktionen unbenutzt (tot). Die erzielten Ergebnisse der dynamischen Codeüberdeckung eignen sich daher sehr gut zur Bereinigung des Quellcodes. Dabei sollte man jedoch zwischen Funktionen unterscheiden, die recht allgemeingültige Aufgaben für Objekte einer Klasse bewältigen können, aber von den instanziierten Objekten unbenutzt bleiben, und jenen Funktionen, die sich nachweislich als Überreste älterer Programmversionen heraus gestellt haben. Letztere können problemlos aus den Quellen entfernt werden, währenddessen Funktionen bzw. Methoden von Klassen mit allgemeingültigen Aufgaben nicht voreilig entfernt werden sollten. In einer späteren Programmversion könnten genau diese Funktionalitäten für instanziierte Objekte benötigt werden.

Zwei Beispiele sollen zur Verdeutlichung dienen. Die Funktion `int TSteering::CallMacro(TMacroTag*)` wird nirgendwo im Programm aufgerufen. Der Funktionskörper ist auskommentiert und als Rückgabewert wird immer TRUE zurückgegeben. Das ist ein ziemlich eindeutiger Hinweis auf eine überflüssige Funktion, die wahrscheinlich aus älteren XCTL-Versionen stammt. Außerdem hat sich heraus gestellt, dass die Ausführung von Makros durch andere Methoden realisiert wird. Diese Funktion könnte demnach problemlos aus den Programmquellen entfernt werden.

Die Funktion `TMatrix TMatrix::operator + (const TMatrix&)` ermöglicht das Addieren zweier Matrizen vom Typ TMatrix. Auch wenn diese Funktion in keiner Programmzeile verwendet wurde, sollte sie nicht aus den Quellen entfernt werden, weil sie eine sehr allgemeingültige und nützliche Aufgabe für Objekte dieser Klasse bereitstellt.

Die Auswertung unbenutzter bzw. toter Funktionen ist jedoch nicht Ziel dieser Arbeit. Die Instrumentierung und die Analyse der dynamischen Codeüberdeckung sollte uns zu Testlücken führen, die im Rahmen der Möglichkeiten geschlossen werden können.

Durch die Analyse der dynamischen Codeüberdeckung konnten sogar Fehler im Programm aufgedeckt werden. Dazu wird an dieser Stelle auf den nächsten Abschnitt verwiesen. Aufrufbare Funktionen, die in keinen Anwendungsfällen vorkommen (Spezifikations-Dokumente) und soweit bekannt von den Physikern nicht verwendet werden, blieben bei der Auswertung unbeachtet. Dazu zählt zum Beispiel das „Justagen-Fenster“ mit all seinen zugehörigen Funktionen.

Tatsächlich ließen sich einige Funktionen aufspüren, die mit den Möglichkeiten der Umgebungssimulation ausführbar sind und mit den definierten Testfällen für die Anwendungsfälle nicht abgedeckt wurden. Selbstverständlich betrifft das die Funktionen, die außerhalb der bekannten Anwendungsfälle liegen, wie zum Beispiel die Benutzung der Zwischenablage. Es wurden aber auch Funktionen aufgefunden, die sich eindeutig in einen der Anwendungsfälle einordnen lassen und mit den definierten Testfällen noch nicht mindestens einmal aufgerufen wurden.

In der folgenden Tabelle sollen diese funktionalen Überdeckungslücken mit Möglichkeiten zur Abdeckung (Testfälle) zusammengefaßt werden.

Unaufgerufene Funktionen	**Möglichkeiten zur Abdeckung**
TCurve::BackStep()	ContinuousScan mit Unterbrechung der Messung. Achtung ! Simulationsalgorithmus beginnt nach einer Unterbrechnung mit der Kurvensimulation erneut.
TPlotData::RenderFormat() TPlotData::RenderAllFormat() TPlotData::DoCopy()	Verwendung der Zwischenablage. Aufruf über „Bearbeiten“ → „Kopieren“.
TCurveShowParam::Dlg_OnHScrollBar()	„Einstellung für die Darstellung“, Scrollbar BildpunkteX/Y bewegen.
dlSetDevice() dMeasureStart() dMeasureStop() dSetExposureValues() dGetExposureValues()	Makrodatei mit Kommando „ChooseDevice“ ausführen.
TAreaScan::rButtonDown()	Rechte Maustaste im Fenster „AreaScan“.
TScan::rButtonDown()	Rechte Maustaste im Fenster „LineScan“.
TCalibratePsd::Dlg_OnHScrollbar()	Im Dialog „PSD Kalibrierung“ Horizontale Scrollbar bewegen.
TChooseScan::Dlg_OnLButtonUp()	Im Dialog „Scan auswählen“ während aktivem AreaScan linke Maustaste drücken.

Unaufgerufene Funktionen	**Möglichkeiten zur Abdeckung**
TAbout::LeaveDialog() TAbout::TAbout() TAbout::Dlg_OnInit() TAbout::Dlg_OnCommand() AboutTheMaker()	Dialog „Über Steuerprogramm“ öffnen und schließen. (Aufruf über „Hilfe“ → „Über“)
SetStatus()	In der INI-Datei unter [Steuerprogram] `ArrangeMode=1` setzen und mit Antrieben arbeiten (z.B. Manuelle Justage). In der Statuszeile wird zusätzlich die Motorstellung in Encoder-Schritten angezeigt.
MenuSelect()	Menüfunktionen im Hauptprogramm direkt auswählen
TPosControl::HScrollbar()	Im Dialog „Verfahren nach Encoder-Position“ horizontaler Scrollbar bewegen. (Funktioniert nicht !!)
TMotor::MoveToPosition()	Im Dialog „Verfahren nach Encoder-Position“ Motor vom Typ TMotor bewegen.
TC_832::SetHome()	Im Dialog „Grundstellung anfahren“ Button „Absolute Null“ für einen C832-Antrieb (IndexLine=1) anklicken.
TC_832::StartLimitWatch() TC_832::StopLimitWatch() LimitWatch()	Im Dialog „Motor-Parameter“ Button „Endschalter“ für einen C832-Antrieb (IndexLine=1) anklicken.
TC_832::IsLimitHit() TC_832::IsIndexArrived() TC_832::StartToIndex()	Im Dialog „Grundstellung anfahren“ Referenzpunktlauf für einen C832-Antrieb (IndexLine=1) anklicken.
TCounterWindow::rButtonDown()	Rechte Maustaste im Fenster „Zähler“.
TSaveDataCmd::TSaveDataCmd() TSaveDataCmd::GetShowData() TSetFileNameCmd::TSetFileNameCmd() TSetFileNameCmd::GetShowData() TChooseDeviceCmd::TChooseDeviceCmd() TChooseDeviceCmd::GetShowData() TSetupScanCmd::TSetupScanCmd() TSetupScanCmd::GetShowData() TControlFlankCmd::ReadyStep() TAreaScanCmd::GetShowData()	Makrodatei mit Kommandos „SaveData“, „SetFileName“, „ChooseDevice“, „SetupScan“ „ControlFlank“ und „AreaScan“ ausführen.

Alle Testfälle sind innerhalb der Umgebungssimulation durchführbar und können weitestgehend mit den Möglichkeiten unseres entwickelten Testsystems automatisiert werden. Mausaktionen, wie das Drücken der linken und rechten Maustaste im Kontext bestimmter Fenster und Dialogboxen, können unter *ATOS* nur mit interaktiven Testskripten durchgeführt werden.

Im Sinne unserer entwickelten Teststrategie sollten die neuen Testfälle so entworfen werden, dass sie gleichzeitig mehrere noch nicht abgedeckte Funktionen auf einmal erfassen. In einigen Fällen ist es sogar vorteilhafter die bereits definierten Testfälle so zu erweitern, dass im Zuge ihrer Durchführung die übersehenen Funktionen aufgerufen werden. Unter diesen Gesichtspunkten bleiben deshalb nur noch zwei neue Testfälle übrig, die dem Regressionstestpaket hinzugefügt werden.

Nach Hinzufügen der neuen und Ergänzung alter Testfälle aus Kapitel 3.3.3, ist die gewünschte minimale Überdeckung, ohne Beachtung unbenutzter Programm-Funktionen, im Rahmen der Umgebungssimulation erreicht.

Vervollständigung des Regressionstestpaketes

Erweiterung der definierten Testfälle

Der Testfall LS.2 wurde so erweitert, dass ein ContinuousScan zunächst gestartet, dann unterbrochen und weitergeführt sowie schließlich abgebrochen wird. Um einen Datenvergleich zu ermöglichen, wird der Scan nochmals ohne Unterbrechung durchgeführt. Ein Aufruf der Funktion `TCurve::BackStep()` ist somit gewährleistet.

Der Testfall DM.1 wurde um die Benutzung der Zwischenablage und der Scrollbars zur Abzissen- und Ordinateneinteilung erweitert. Obwohl die Zwischenablage nur von externen Bitmap-Verarbeitungsprogrammen eingelesen werden kann, wird zumindest der erfolgreiche Aufruf der Funktion innerhalb des XCTL-Systems überprüft. Abgedeckt werden nun `TPlotData::RenderFormat()`, `TPlotData::RenderAllFormat()`, `TPlotData::DoCopy()` und `TCurveShowParam::Dlg_OnHScrollBar()`.

Im Testfall MS.1 wird zusätzlich ein Testmotors vom Typ TMotor eingesetzt. Die Funktion `TMotor::MoveToPosition()` findet somit in einem Testfall ihren Aufruf.

Im Testfall ARS.3 wird zusätzlich die horizontale Scrollbar bei der Kalibrierung des PSD's benutzt. Somit wird nun auch die Funktion `TCalibratePsd::Dlg_OnHScrollbar()` mindestens einmal aufgerufen.

Für den Testfall AS.1 wurde die präparierte Makrodatei um einige Kommandos erweitert, um folgende Funktionen abzudecken: `dlSetDevice()`, `dMeasureStart()`, `dMeasureStop()`, `dSetExposureValues()`, `dGetExposureValues()`, `TSaveDataCmd::TSaveDataCmd()`, `TSaveDataCmd::GetShowData()`, `TSetFileNameCmd::TSetFileNameCmd()`, `TSetFileNameCmd::GetShowData()`, `TChooseDeviceCmd::TChooseDeviceCmd()`, `TChooseDeviceCmd::GetShowData()`, `TSetupScanCmd::TSetupScanCmd()` und `TSetupScanCmd::GetShowData()`. Die Funktion `TControlFlankCmd::ReadyStep()` wird beim Kommando `ControlFlank` aufgerufen (Topographie) und dient nur zur Ausgabe einer Zeichenkette in das Reportfenster der Makroausführung. Dieses Kommando wird im Testfall nicht ausgeführt, da es zur Festlegung der Topographie-Parameter keinen Makrobefehl gibt und die Topographie schon in einem anderen Testfall überprüft wird. Genauso verhält es sich mit dem Kommando `AreaScan`, mit dem ein AreaScan aus einer Makrodatei gestartet werden kann. Eine Möglichkeit zur Definition der Parameter über das Makrofile gibt es nicht, weshalb auch auf den Einsatz dieses Kommandos im Testfall verzichtet wurde und die Funktion `TAreaScanCmd::GetShowData()` zur Intensitätsausgabe in der Statuszeile unaufgerufen bleibt.

Definition neuer Testfälle

ID: AE.2
Name: Aufrufen der Dialogbox „Über Steuerprogramm“
Kurzbeschreibung:
Die Dialogbox „Über Steuerprogramm“ wird aufgerufen. Die Adressen der Entwickler werden über die entsprechenden Buttons aufgerufen und angezeigt.

ID: MS.3
Name: Spezielle Arbeiten mit C832-Antrieben
Kurzbeschreibung:
Die Dialogboxen „Grundstellung anfahren“, „Motor-Parameter“ und „Manuelle Justage“ werden benutzt. Der Antrieb Omega wird auf seinen absoluten Nullpunkt zurück gesetzt, auf seine Indexposition und schließlich auf seine Grundstellung gefahren. In der Dialogbox „Motor-Parameter“ wird der Antrieb Theta ausgewählt und seine Endschalter-Funktion aktiviert. Um die Auslösung des Endschalters zu testen, muss der Antrieb über eine seiner Schranken gefahren werden. Das wird erreicht, indem die Absolute Null sehr weit rechts gesetzt und der Antrieb über die rechte Schranke gefahren wird.

Ausnahmen

Zur Ergänzung der definierten Testfälle aus den Ergebnissen der dynamischen Codeüberdeckung muss man einen Kompromiß zwischen Testaufwand und dem erzielten Nutzen finden. Sicherlich wäre es möglich, Testfälle zu definieren, die den Tester dazu auffordern, in jedem Fenster die rechte Maustaste zu drücken, um einmal die Funktionen zur Darstellung der Kontextmenüs aufzurufen. Dabei ist jedoch zu beachten, dass jeder interaktive Testfall den automatisierten Anteil des Regressionstestpaketes vermindert. Auf die Überprüfung dieser Funktionen wird daher zugunsten der Automatisierung verzichtet. Die Funktionen `TAreaScan::rButtonDown()`, `TScan::rButtonDown()`, `TChooseScan::Dlg_OnLButtonUp()` und `MenuSelect()` werden aus diesem Grund durch keinen der Testfälle aufgerufen.

Die Funktion `SetStatus()` wird ebenfalls nicht innerhalb eines Testfalles aufgerufen, da die Statuszeile in der aktuellen Programmversion nicht automatisiert ausgelesen werden kann. Ein Testfall zur Bewegung der horizontalen Scrollbar in der Dialogbox „Verfahren nach Encoder-Position“ ist nicht möglich, weil diese Funktion auf Seiten des XCTL-Programmes fehlerhaft ist.

5.1.3 Fehler

Bei der Suche nach Gründen für nicht abgedeckte Funktionen stellte sich die Ursache häufig als Programmierfehler heraus, die mit einer statischen Quellcode-Analyse wahrscheinlich niemals entdeckt worden wären. Die folgenden Fehlerbeschreibungen sind den Dokumenten im Web-Repository hinzugefügt worden.

1. Fehlende Destruktoraufrufe

Objekte, die mittels `new` instanziiert wurden, sollten am Ende ihrer Benutzung mittels `delete` gelöscht werden, um reservierten Speicherplatz wieder frei zu geben.

UseCase „Motorsteuerung“

Nr.	Datum	Status	Wer?	Beschreibung

16	08.08.2002
	21.10.2002
		1ST
		FIX
			hanisch
			hanisch
				Charakterisierung:
				Das Objekt der Klasse `TMList`
				wird in den Methoden
				`BOOL DllEntryPoint(HANDLE,DWORD,LPVOID)` bzw.
				`int WINAPI LibMain(HINSTANCE,WORD,WORD,LPSTR)`
				erzeugt aber nirgendwo gelöscht.
				(`lpMList=(LPMList) new TMList(10,NULL);`)
				Ursache:
				Fehlendes `delete lpMList;`
				Behebung:
				Löschen des Objekts in der Methode
				`int CALLBACK WEP(int)` mit
				`delete lpMList;`
				(Datei `M_LAYER.CPP`)

Subsystem „Interne Funktionen“

Nr.	Datum	Status	Wer?	Beschreibung

1	08.08.2002	1ST	hanisch	**Charakterisierung:**
				Das Objekt der Klasse `TDataBase`
				wird in den Methoden
				`BOOL DllEntryPoint(HANDLE,DWORD,LPVOID)` bzw.
				`int WINAPI LibMain(HINSTANCE,WORD,WORD,LPSTR)`
				erzeugt aber nirgendwo gelöscht.
				(`lpDataBase=(LPDataBase)new TDataBase();`)
				Ursache:
				Fehlendes `delete lpDataBase;`
				Behebung:
				Löschen des Objekts in der Methode
				`int CALLBACK WEP(int)` mit
				`delete lpDataBase;`
				(Datei `L_LAYER.CPP`)
				
				Achtung! Scheinbar enthält der
				Destruktor ∼`TDataBase()` bzw.
				`TDataBase::New()` einen Fehler,
				weshalb `delete lpDataBase;` unter
				Windows 3.11 zum Absturz beim
				Beenden des Programmes führt.

UseCase „Ablaufsteuerung“

Nr.	Datum	Status	Wer?	Beschreibung

4	08.08.2002
	21.10.2002
		1ST
		FIX
			hanisch
			hanisch
				Charakterisierung:
				Das Makro-Kommando wird direkt aufgerufen aus der Funktion `TTopographyExecute::Dlg_OnCommand()`. Dabei wird das Objekt der Klasse dynamisch erzeugt. (`TheCmd=(TControlFlankCmd*) new TControlFlankCmd(Cmd);`). Das zuletzt ausgeführte Kommando-Objekt wird beim Programmende nicht gelöscht.
				Ursache:
				Löschen der Kommando-Objekte geschieht immer nur vor der Erzeugung eines neuen Kommando-Objekts.
				Behebung:
				Löschen des Kommando-Objekts im Destruktor von `TSteering`. Diesen Destruktor gibt es nicht ! Einfügen eines Destruktors für die Klasse `TSteering`:
				
				`TSteering::~TSteering(void) {`
				`if (TheCmd) delete (TheCmd);`
				`}`

2. Korrekturpolynom

Die Umrechnung zwischen internen Motorschritten (Encoder-Schritte) und Winkelschritten wird mit den Methoden `TMotor::Translate(long& ,double)` bzw. `TMotor::Translate(double&, long)` berechnet. Dabei kann die Umrechnung entweder linear oder mit einem sogenannten Korrekturpolynom vorgenommen werden. Welche der beiden Varianten für einen Antrieb eingesetzt wird, legt man in der Datei `HARDWARE.INI` fest. Hier Bestimmen die Parameter `Correction, Koeff_1, Koeff_2 und Koeff_3` die Verwendung einer linearen bzw. polynomialen Umrechnung.

UseCase „Motorsteuerung“

Nr.	Datum	Status	Wer?	Beschreibung
17	08.08.2002	1ST	hanisch	**Charakterisierung:** Die Methoden zur polynomialen Umrechnung `TMotor::funcd()` und `TMotor::rtsave()` werden niemals aufgerufen, obwohl in der INI-Datei die jeweiligen Parameter dafür gesetzt sind. **Ursache:** Diese Methoden werden nur aufgerufen, wenn gilt `eCorrStatus=CorrPolynom` in `TMotor::Translate(long&,double)`. An dieser Stelle ist jedoch immer `eCorrStatus=CorrLinear`, unabhängig von den Einstellungen in der INI-Datei. **Behebung:** Hierfür ist eine umfangreiche Analyse des Quellcodes nötig. Außerdem müssen Rücksprachen mit den Physikern gehalten werden, um die Bedeutung der Umrechnungsarten in Erfahrung zu bringen. Die Dokumente im Web-Repository reichen zur Klärung dieser Problematik nicht aus.

5.2 Einsatz in der Praxis

Dass sich das Testsystem *ATOS* schon mehrfach in der Projektarbeit bewährt hat, sollen die folgenden Beispiele demonstrieren. In den letzten Wochen unterlag das XCTL-System massiven Veränderungen, sowohl auf der syntaktischen Seite (Programmdesign), als auch auf der semantischen Seite (Programmfunktionalitäten). Regressionstestläufe waren zur Bewahrung des Programmverhaltens unabdingar, wofür schon weit vor der eigentlichen Fertigstellung der Testsuite *ATOS* alle definierten Testfälle automatisiert und regelmäßig durchgeführt wurden. Tatsächlich konnten mit Hilfe der für die Umgebungssimulation spezifizierten Testsequenzen einige Fehler erfolgreich aufgedeckt und umgehend behoben werden.

Skript:	`Test_ARS.2.HTS`
Datum:	07.06.2002
Fehler:	Das XCTL-System stürzt beim Aufruf eines AreaScans mit einem 0-dimensionalen Testdetektor (SLD-Scan) ab.
Ursache:	Der Fehler wurde von der Projektgruppe zur Überarbeitung der Detektorkomponente lokalisiert und entfernt.

Skript:	`Test_TP.1.HTS`
Datum:	28.06.2002
Fehler:	Im Dialog „Einstellungen Topographie“ ist keine Auswahl von Antrieb „DF“ möglich.
Ursache:	Fehler bei Zugriff auf Index 0 in Schleife `while (++i<mlGetAxisNumber())` in der Methode `TTopographySetParam::Dlg_OnInit()`.

Skript:	`Test_AS.1.HTS`
Datum:	28.06.2002
Fehler:	Einlesen der Datei `SCAN.MAK` schlägt fehl.
Ursache:	Makrokommando `ChooseDevice` heißt jetzt `ChooseDetector` und wurde nicht in `SCAN.MAK` geändert.

Skript:	`Test_AE.1.HTS`
Datum:	28.06.2002
Fehler:	Die Eingabefelder Substrat und Name wurden bei der Überarbeitung der Dialogbox „Allgemeine Einstellungen“ vertauscht, weshalb der Vergleich mit *DataDiff* zwischen der Datei `DEVELOP.INI` und der Solldatei `DEVELOP.INI.REF` scheitert.
Ursache:	IDs für die Eingabefelder in der Datei `MAIN.RC` waren vertauscht.

Skript:	`Test_D0.1.HTS`
Datum:	06.08.2002
Fehler:	Die Aktualisierung des Zählerfensters bleibt stehen, sobald der Dialog zur Parameter-Einstellung geschlossen wird.
Ursache:	Der Fehler wurde von der Projektgruppe zur Überarbeitung der Detektorkomponente lokalisiert und entfernt.

Skript:	`Test_AJ.1.HTS`
Datum:	23.09.2002
Fehler:	Bei den Motorstellungen DF=5.0, TL=-20.0, CC=140.0 hat der 0-dimensionale Testdetektor „Counter“ eine Intensität von ca. 15917, erwartet wird aber ca. 22554.
Ursache:	Der Fehler trat nach der Überarbeitung des 0-dimensionalen Testdetektors (auf Grundlage der Datenbasis `TESTDEV.DAT`) auf.

Skript:	`Test_ARS.1.HTS`
Datum:	23.09.2002
Fehler:	Die Messwerte des 1-dimensionalen Testdetektors „PSD“ werden in umgekehrter Reihenfolge in die Ausgabedateien (`*.psd`) ausgeschrieben, weshalb der Vergleich mit *DataDiff* zwischen den Istdateien (`*.psd`) und der Solldatei (`*.psd.ref`) scheitert, obwohl der Unterschied optisch kaum wahrnehmbar ist.
Ursache:	In alten Programmversionen des XCTL-Systems wurde die im Dialog „Einstellungen für den PSD“ festgelegte Auslesereihenfolge für simulierte Intensitätswerte ignoriert.

5.3 Verwandte Arbeiten

Die Realisierung eines Testsystems, das die Durchführung skriptbasierter Testfälle über die grafischen Benutzerschnittstellen des Testobjektes vornimmt, ist in der Praxis ein gängiges Verfahren für BlackBox-Tests. Die Gewinnung eines ausführbaren Skriptes erfolgt dazu in modernen Testwerkzeugen mit Hilfe eines Makrorekorders, der alle durchgeführten Aktionen auf den Oberflächen-Elementen der zu testenden Anwendung aufzeichnet und als Skriptkommandos in einer Datei abspeichert. Die generierten Skripte der mächtigen Testsuite *WinRunner* [24] von *Mercury Interactive* weisen im kontextsensitiven Modus sogar eine augenscheinliche Ähnlichkeit zu unserer Skriptsprache auf. Hierfür einige Beispiele:

WinRunner - Skriptkommando	**HTS - Skriptkommando**
`button_press("Ok");`	`ACTION,"...",BUTTON,CLICK,"Ok"`
`edit_get_text("name:", text);`	`READ,"...",EDITBOX,TEXT,text,"name:"`
`win_exists("...");`	`WINDOWSEXISTS,"...",YES`
`invoke_application("notepad.exe");`	`LAUNCH,ABS,"notepad.exe","",NOWAIT`
`menu_select_item("File:Open");`	`ACTION,"...",MENU,CLICK,"File","Open"`

Der Aufwand zur Implementierung eines Testsystems mit Makrorekorder würde den erwarteten Nutzen in unserem Fall nicht rechtfertigen, was uns dazu bewogen hat, ausführbare Testskripte auf anderen Wegen zu gewinnen.Mit einer definierten Sprache zur Beschreibung von Aktions- und Aus-

wertungsschritten, ist die Quelle von Skriptdateien nicht bestimmt. Vielmehr kann die Skriptsprache als Schnittstelle zwischen anderen Testwerkzeugen wie beispielsweise dem *Classification Tree Editor* und unserem entwickelten Testsystem dienen.

Ein ähnliches Verfahren wird in der Diplomarbeit von Stefan Lützkendorf [10] eingesetzt. Bis auf die Idee zur automatisierten Gewinnung von Testskripten aus den Attributen der Elemente eines Klassifikationsbaumes (siehe Abschnitt 3.8.1) gibt es jedoch keine weiteren Gemeinsamkeiten. Das generierte Skript dient hierbei als Grundlage zur Erzeugung und Einbindung von Quellcode in das untersuchte Testobjekt. Die Testdurchführung wird demnach vom Prüfling selbst vorgenommen, was sich grundsätzlich von unserem Ansatz eines von außen gesteuerten BlackBox-Testes unterscheidet. Auch die Auswertung der Ausgaben folgt einer anderen Strategie. Während in unserem Verfahren jeder Testschritt zur Laufzeit überprüft wird, erfolgt der Vergleich zwischen Ist- und Solldaten beim Verfahren von Herrn Lützkendorf erst nach Beendigung des Testdurchlaufs auf Grundlage der angelegten Logdateien. Die Logdateien unseres Testsystems protokollieren im Gegensatz dazu den Erfolg aller ausgeführten Skriptkommandos und dienen ausschließlich der Lokalisierung einer möglichen Fehlerquelle.

Der Einsatz proprietärer Skriptsprachen zur Spezifikation von Testfällen findet man sehr häufig in der Praxis. Die Firma *Vossloh System-Technik GmbH* entwickelt Testsysteme, die eine automatisierte Durchführung von Funktionstest für Anwendungen aus dem Bereich Telekommunikation ermöglichen. Die automatischen Testabläufe werden durch Testprofile definiert. Diese enthalten Kommandos zur Erzeugung von Stimuli an den Prüfling, zur Bewertung der Prüflingsreaktionen und zur zeitlichen Ablaufkontrolle des Tests. Durch Verwendung dieser Skriptsprache, die auf eine effiziente Ausführung in Echtzeit zugeschnitten ist, lassen sich neue, anwendungsspezifische Testkommandos schnell implementieren. Das entspricht unserer Idee zur Abbildung von HTS-Kommandos auf Kommandos der betriebssystemnahen Skriptsprache ATS (Abschnitt 3.5.1).

Auch bei *Vossloh System-Technik GmbH* entstand schnell der Wunsch nach einer intuitiveren, graphischen Eingabenotation. Zusammen mit der Medizinischen Hochschule Lübeck entstand ein Projekt zur Entwicklung eines graphischen Front-Ends zur Testfalldefinition. Als Notation werden „Message Sequence Charts“ (MSC) verwendet, die den „Sequence-Diagrams“ der objektorientierten Modellierung UML entsprechen. Grundgedanke ist es, zu jedem Sprachkonstrukt der Skriptsprache eine Darstellung durch MSCs zu

finden, so dass eine automatische Umsetzung der MSC in Testskripte möglich ist. Das Ziel, Testskripte aus einem graphischen Editor heraus automatisiert zu gewinnen, wird in unserer Arbeit mit der Attributierung von Klassifikationsbäumen verfolgt. Das Verfahren der Firma *Vossloh System-Technik GmbH* wurde bei einem Treffen der GI-Fachgruppe 2.1.7 im Mai 2000 vorgestellt (siehe [14, S.2 ff]).

Auf der selben Tagung wurde eine Diplomarbeit [15, S.5 ff] zum Thema „Testapplikation im Bereich der ISDN-Kommunikation über eine CAPI-Schnittstelle“ vorgestellt. Der Student entwarf dafür eine Skriptsprache, deren Syntax unserer HTS- bzw. ATS-Sprachspezifikation auffallend ähnlich ist. Ein Skript besteht wie bei uns aus einer Folge von Befehlszeilen oder Schleifen. Befehlszeilen setzen sich aus einem eindeutigen Befehlsnamen und einer anschließenden Liste von Parametern zusammen. Die Auswertung beim automatischen Test wird durch eine Logdatei unterstützt. Bei einem Systemabsturz kann man damit z.B. verifizieren, an welcher Stelle das System abgestürzt ist.

Beiden proprietären Skriptsprachen ist gemeinsam, dass ihre Kommandos für einen speziellen Anwendungsbereich entworfen worden sind. Unsere Skriptsprache operiert auf der Ebene von Oberflächen-Elementen einer Windowsapplikation, womit sie erheblich flexibler ist, solange die Testobjekte über eine grafische Benutzerschnittstelle verfügen.

Die Klassifikationsbaum-Methode zum systematischen Entwurf von Testfällen hat sich in den letzten Jahren immer mehr durchgesetzt. Nicht zuletzt trug die Möglichkeit zur Kopplung mit anderen Testwerkzeugen dazu bei. Das Testsystem *TESSY* der DaimlerChrysler AG kann Diagramme des *Classification Tree Editor* einlesen, um daraus Eingabedaten für die Schnittstellen eines Testobjektes (Funktionen eines C-Programmes) automatisch zu generieren. Auf einer Konferenz der GI-Fachgruppe 2.1.7 im Oktober 2001 wurde der Einsatz der Klassifikationsbaum-Methode beim „Time Partition Testing“ (TPT) vorgestellt (siehe [17, S.6/7]). TPT ist ein spezialisiertes Testverfahren für den automatischen Funktionstest des kontinuierlichen Verhaltens eingebetteter Systeme, wobei jeder Testfall mit hierarchischen, hybriden Automaten modeliert wird. Hierbei wird ein Testfall in eine Folge von einzelnen Zeitphasen zerlegt, deren Übergänge an Transitionsbedingungen geknüpft sind. Der grundsätzliche Ablauf einzelner Phasen des Tests ist für alle Testfälle gleich. Das Verhalten innerhalb der Phasen unterliegt jedoch Variationen, die sich mit der Klassifikationsbaum-Methode hervorragend modellieren lassen.

Um den Prozeß der automatisierten Testfall-Generierung zu unterstützen, sollen Spezifikation von logischen Abhängigkeiten zwischen den Klassen eines Klassifikationsbaumes im *CTE XL* (Classification Tree Editor eXtended Logics) möglich gemacht werden. Beim Treffen der GI-Fachgruppe 2.1.7 im Februar 2001 wurden die Arbeiten an dieser Versionserweiterung von Mitarbeitern der DaimlerChrysler AG vorgestellt (siehe [16, S.6]). Erreicht werden sollen dabei vorrangig zwei Ziele. Zum einen soll die Kombination von widersprüchlichen Klassen in Testfällen verhindert werden und zum anderen sollen Testfälle automatisch vervollständigt werden, wenn sich aus den vom Benutzer in einem Testfall selektierten Klassen und den bestehenden logischen Regeln die Auswahl weiterer Klassen eindeutig ableiten läßt.

Das Konzept, Anwendungen und Komponenten über grafischen Benutzerschnittstellen zu testen, gewinnt zunehmend an Bedeutung. In einer Dissertation von Atif M. Memon [4] wird ein plattformunabhängiges Framework vorgestellt, welches die grundsätzlichen Aufgaben eines Testsystems wie Testfallgenerierung, Testdurchführung und Testauswertung erfüllen soll. Zudem kann eine Komponente des Frameworks die Durchführung von Regressionstests anbieten und sogar die Anpassung (Reparatur) der Testfälle auf neue Situationen automatisieren. Dabei verweist der Autor auf verschiedene verwandte Arbeiten, wie beispielsweise das *visual test development environment (TDE)* [5] und kritisiert die Fülle der manuell durchzuführenden Aufgaben, die mit der Benutzung solcher Capture&Replay-Werkzeuge [6, 7] verbunden sind.

Grundlage dieser Arbeit bildet die Repräsentation einer beliebigen GUI in Form von Objekten mit ihren Eigenschaften. In einer Dialogbox bilden zum Beispiel alle Buttons mit ihren Eigenschaften wie Beschriftung, Position und Schriftfarbe ein Objekt. Diese Objekte müssen zunächst vom Testdesigner manuell oder automatisiert aufgenommen werden (z.B. mit Hilfe der Ressourcen-Editoren aus den Entwicklungsumgebungen wie von Borland). Das kann beliebig genau bzw. detailliert vorgenommen werden. Es können auch nur die Eigenschaften zusammengestellt werden, die für den Testdesigner von Interesse sind, wobei davor gewarnt wird, dass evt. ungewünschte Änderungen an der Oberfläche während der Testdurchführung dann unentdeckt blieben.

Außerdem werden sogenannte GUI-Operatoren entworfen. Jeder GUI-Operator besteht aus einem Namen, einer Menge von Vorbedingungen, und einer Menge von Effekten. Vorbedingungen und Effekte beziehen sich dabei nur auf Eigenschaften, die vom Testdesigner in der GUI-Repräsentation aufgenommen wurden. Die gesamte GUI bzw. das Testobjekt lässt sich

dann in Form eines Graphen mit Knoten (GUI-Zustände) und Kanten (GUI-Operationen) beschreiben. Jede Operation führt die GUI in einen neuen Zustand, in dem ihre Eigenschaften verändert wurden. Dabei kann nicht jeder beliebige Operator auf einen Ausgangszustand angewendet werden. Die Hintergrundfarbe eines Fensters kann zum Beispiel nur dann geändert werden, wenn dieses auch offen ist. Solche Einschränkungen werden durch die Vorbedingungen in den GUI-Operatoren sichergestellt.

Liegt einmal eine solche Definition der zu testenden Oberfläche in Form eines Graphen vor, können verschiedene Algorithmen die geforderten Testaufgaben leicht automatisiert werden. So können beispielsweise Testfälle aus der Angabe von Anfangszuständen und den gewünschten Endzuständen generiert werden. Ein Algorithmus, der aus dem Forschungsgebiet der künstlichen Intelligenz [8, 9] übernommen wurde, wählt dann eine Menge von möglichen Wegen (Pfaden) aus dem Graphen und bietet sie als abstrakte Testpläne an. Dabei gibt es häufig mehrere Wege zum Erreichen des Endzustandes, was bei der automatisierten Generierung von Testplänen bzw. Testfällen durchaus erwünscht ist.

Eine Komponente, die in der Arbeit leider nicht näher erläutert wird, sorgt in einem letzten Schritt für die eigentliche Abbildung dieser Testpläne in Form von sogenannten GUI-Events auf physische Aktionen (Maupositionierungen/-aktionen und Tastaturaktionen). Diese Komponente kann für jede beliebige Plattform entwickelt werden, was das vorgestellte Framework sehr flexibel macht.

Auch die Erwartungswerte (Zustände der Oberfläche) lassen sich auf Grundlage der GUI-Repräsentation und der spezifizierten Operatoren automatisiert entwickeln. Aus einem beliebigen Zustand und einer Folge von angewendeten Operationen, lässt sich auf den Zielzustand schließen. Das ist der Grund, warum sich das vorgestellte Konzept schwierig auf unser Testobjekt (XCTL-System) anwenden lässt. Die Funktionen des XCTL-Systems sind sehr eng mit den grafischen Elementen der Oberfläche gekoppelt. Eingaben in den Feldern einer Dialogbox führen zu Ausgaben, deren Inhalt abhängig von sehr vielen komplexen und zusammenhängenden Faktoren des XCTL-Systems ist. Eine Messung der Halbwertsbreite aus der Dialogbox zur Manuellen Justage stößt beispielsweise eine ganze Reihe von Motor- und Detektorfunktionen an, die schließlich zu einem berechneten Messwert nach einer bestimmten Zeitdauer führen. Wenn man den berechneten HWB-Wert mit einem Erwartungswert vergleichen möchte, müsste ein entsprechender GUI-Operator für die Repräsentation genauso komplex sein, wie die programminternen Funktionen. Dazu ist es mit dem Framework möglich, einen

GUI-Operator um prozedurale Elemente zu erweitern. Die Problematik der Abbildung von Programmoperationen auf äquivalente GUI-Operatoren wird auch in der Dissertation angesprochen.

Dass sich die Grundkonzepte unserer Arbeit von denen der Dissertation unterscheiden, wird beim Vorgehen zur Analyse der Testfall-Vollständigkeit deutlich. Währenddessen wir mit dem Entwurf der Testfälle auf eine ausreichende Überdeckung aller aufrufbaren Programmfunktionen abzielen, wird in der Arbeit von Herrn Memon sogar von einer quellcodebasierten Überdeckungsmethode abgeraten. Grafische Oberfläche und Quellcode eines Programmes liegen auf verschiedenen Abstraktionsstufen, weshalb man bei einer ausreichenden Quellcodeüberdeckung noch nicht ein korrektes Verhalten der GUI schließen darf. Beim Testen einer GUI steht die Korrektheit der Interaktionen zwischen dem System und dem Benutzer im Vordergrund, die mit der neu entwickelten eventbasierten Überdeckungsanalyse sichergestellt werden soll. Von der GUI-Repräsentation (Graph) ausgehend erhält man eine ausreichende Überdeckung, wenn möglichst viele Pfade in diesem Graphen durchlaufen wurden, d.h. es müssen möglichst viele und lange Sequenzen mit GUI-Operationen ausgeführt werden.

Wir benutzen die grafische Oberfläche des XCTL-Systems als Schnittstelle zum Test der darunterliegenden Programmfunktionen im konventionellen Sinne, weshalb unsere entwickelte Skriptsprache bei weitem nicht die Möglichkeiten zur Abfrage aller Zustände eines grafischen Oberflächenobjektes bietet. Ferner wären Kommandos zum Umgang mit Dateien (Kopieren, Löschen) und zum Aufruf von externen Programmen für einen Test, der nur auf Ebene der grafischen Benutzerschnittstelle stattfindet, nicht notwendig, was den Unterschied beider Ansätze unterstreicht.

5.4 Zusammenfassung und Ausblick

5.4.1 Selbsteinschätzung

Das Ziel, häufig anfallende Regressionstests zu automatisieren, konnte mit den Ergebnissen dieser Arbeit zufriedenstellend erreicht werden. Maßgeblich dafür, war der Einsatz einer benutzerfreundlichen Skriptsprache und die Entwicklung verschiedener Möglichkeiten zur Unterstützung beim Entwurf von Skriptdateien. Da wir uns hauptsächlich mit der Frage beschäftigt haben, wie eine Windowsapplikation generell über seine grafischen Benutzerschnittstellen getestet werden kann und nicht welches spezielle Testobjekt zugrunde

liegt, ist ein sehr flexibles Testsystem entstanden, das auch in anderen Projekten seinen Einsatz finden könnte.

Im Bereich der Möglichkeiten, die durch die simulierten Detektoren und Motoren gegeben sind, haben wir ein breites Spektrum der Programmfunktionalitäten mit den entwickelten Testsequenzen abdecken können. Trotz aller Schwierigkeiten, die bei der dynamischen Überdeckungsanalyse mittels Instrumentierung des XCTL-Systems auftraten, war das Ergebnis zur Bewertung der Vollständigkeit der Testfälle sehr hilfreich. Mit erfolgreichen Durchläufen eines zusammengestellten Regressionstestpaketes kann jedoch nicht die Korrektheit des Programmes beim Einsatz an realen Messplätzen gewährleistet werden. Vielmehr können schon während des Reengineering-Prozesses Fehler vermieden werden, die das ursprüngliche Verhalten der Steuerungssoftware unter Laborbedingung verändern könnten. Die fehlerfreie Durchführung aller Testsequenzen innerhalb der Umgebungssimulation führt somit mit hoher Wahrscheinlichkeit Programmversionen, die sich auch in Umgebung realer Messapparatur wie erwartet verhalten.

Beim Design des Testsystems legten wir großen Wert auf Möglichkeiten zur Anpassung auf spezielle Anwendungsfälle. Durch das zweischichtige Skriptsprachen-Modell lassen sich bestehende Kommandos relativ mühelos in ihrem Verhalten verändern oder sogar neue Kommandos definieren. Dem Skriptentwickler bietet sich ferner die Möglichkeit, beliebige Programme für den Einsatz in einem Testfall heranzuziehen. So ist man beispielsweise beim Dateivergleich nicht auf *DataDiff* angewiesen und könnte stattdessen ein anderes Werkzeug zu Rate ziehen. Testschritte, die mit den Mitteln der Skriptsprachen nicht oder schwer automatisierbar sind, können durch den Einsatz interaktiver Dialoge mit dem Tester durchgeführt werden.

Das Problem der Abhängigkeit von numerischen IDs zur Ansteuerung der Oberflächen-Elemente über Windows-Nachrichten konnte erfolgreich mit dem entwickelten *RCParser* gelöst werden. Jedes Oberflächen-Element des Testobjektes, ob Menü, Dialogbox, Fenster oder Control wird mit einem eindeutigen und verständlichen Bezeichner in einer zentralen Datenbank verwaltet.

Die Klassifikationsbaum-Methode erbrachte hingegen nicht den erwarteten Nutzen. Zustandsorientierte Klassifikationsbäume zu attributieren ist eine sehr mühsame und fehleranfällige Aufgabe. Das Verfahren eignet sich, wenn man eine Menge von Testsequenzen für eine untersuchte Komponente übersichtlich und versteckt in einer einzigen Datei (CTE-Diagramm) verwal-

ten möchte. Bei Bedarf kann das Diagramm zur Generierung von Testskripten aus seinen Attributen herangezogen werden.

Aktionsorientierte Diagramme lassen sich leichter attributieren, da ihre Elemente bestimmte Operationen auf dem Testobjekt beschreiben und direkt in Skriptkommandos übersetzt werden können. Das Problem bei diesem Verfahren liegt eher bei der Konstruktion der Testsequenzen über die Kombinationstabelle. Viele Testschritte sind voneinander abhängig. Die Zeitdauer für eine Messung über einen bestimmten Winkelbereich ist beispielsweise von vielen Parametern, die über Dialogboxen und Konfigurationsdateien mit früheren Testschritten festgelegt werden, abhängig. Die Klassifikationsbaum-Methode kann daher nur als Unterstützung beim systematischen Entwurf von Skriptdateien dienen. Eine automatisierte Generierung sinnvoller Testskripte durch Auswahl willkürlicher Kombinationen kann man nicht erwarten.

5.4.2 Ausblick

Aufgrund der flexiblen Gestaltung des Testsystems und seiner Komponenten bieten sich viele Möglichkeiten zur Verbesserung und Erweiterung. Denkbar wäre der Entwurf einer Komponente zum approximativen Vergleich von Bildschirmgrafiken (Bitmaps). In unserem Fall werden Kurvengrafiken als Liste von Punkten in Text-Dateien abgespeichert und eingelesen. Zur Laufzeit des Steuerungsprogrammes wird mit diesen Daten ein Bitmap erstellt und auf den Bildschirm ausgegeben. Binäre Grafikdateien, die sich für einen Vergleich mit Solldateien eignen, werden hingegen nicht erzeugt. Ein Bitmap-Vergleich mit Grafiken aus den darstellenden Fenstern heraus ist ein sehr schwieriges Unterfangen. Die Entwicklung einer solchen Komponente für Regressionstests in der Umgebung des XCTL-Systems ist aber auch nicht unbedingt notwendig, da der Vergleich von Kurvengrafiken auf Ebene der Messwertdateien (ASCII-Text) problemlos vorgenommen werden kann.

Genügt das Vergleichsprogramm *DataDiff* nicht den geforderten Ansprüchen, kann eine neue Komponente entwickelt werden, die völlig unabhängig von der Testsuite *ATOS* ist. Man könnte sich vorstellen, Metainformationen wie Toleranzwerte von den eigentlichen Solldaten zu trennen. Für den Dateivergleich mit einer solchen Testkomponente wären folglich zwei Referenzdateien bereitzustellen: eine Solldatei und eine Datei mit Informationen über Struktur und Behandlung der in der Solldatei enthaltenen Daten. Vorteile bringt diese Strategie nur dann, wenn viele Solldateien mit gleichem strukturellen Aufbau auf die selben Metainformationsdateien zugreifen können. Eine redundante und fehleranfällige Präparierung jeder Solldatei wie bei *DataDiff* könnte somit vermieden werden.

Sehr viele Ausgaben und Statusinformationen des XCTL-Systems erfolgen über seine Statuszeile. Eine ungünstige Software-Architektur verhindert den Zugriff mit den Möglichkeiten unseres Testsystems. Grundlage der Statuszeile bildet eine Grafik (Bitmap), die bei jeder Aktualisierung überschrieben und angezeigt wird. Für einen effektiven Einsatz des Testsystems auf ein bestimmtes Testobjekt sollte es bestimmten Richtlinien bei der Implementierung seiner grafischen Benutzerschnittstelle entsprechen. Vielleicht gelingt es im Zuge des Reengineerings, die Statusinformationen mit Hilfe eines standardisierten Oberflächen-Elementes auszugeben, so dass sie mit entsprechenden HTS-Kommandos ausgelesen und ausgewertet werden können.

Die definierten Testfälle zusammen mit den ergänzenden Testfällen, die aus dem Ergebnis der Vollständigkeitsanalyse entstanden sind, überprüfen die Funktionalitäten einer Programm-Version vom 20.01.2002. Im Zuge der Projektarbeit ist das XCTL-System inzwischen um eine Funktion zur Automatisierung von Protokollierungsaufgaben erweitert worden. In naher Zukunft wird die Dialogbox zur „Manuellen Justage“ durch eine wesentlich komfortablere Dialogbox ausgetauscht. Für die jeweiligen Anwendungsfälle sind von den Entwicklergruppen bzw. von einem dedizierten Tester entsprechende Testskripte zu entwerfen und dem Regressionstestpaket hinzuzufügen. Diese Beispiele zeigen, dass ein erfolgreicher Einsatz des Testsystems zukünftig im hohen Maße von der Disziplin der Projektgruppen abhängig ist. Für jede neue Programmfunktion ist eine Testsequenz zu entwerfen. Jede Veränderung am Verhalten des Programmes bedingt die Überarbeitung und Anpassung bereits definierter Testfälle.

Die Abfrage von Statusinformationen einiger Oberflächen-Elemente ist mit den Möglichkeiten der entwickelten Skriptsprache ATS leider nicht durchführbar. Ein Kommando zur Abfrage des Aktivierungszustandes eines Menüpunktes in der Skriptsprache HTS (`TEST,"...",MENU,CHECKED,"..."`) erfordert den Zugriff auf eine komplexe Speicherstruktur, die als Ergebnis einer Windows-Nachricht zurückgegeben wird (siehe Abschnitt 2.5.2). Für solche Fälle sind die Kommandos der Skriptsprache ATS nicht ausgelegt. Wenn auf Abfragen dieser Art nicht verzichtet werden kann, müssen entweder einschlägige Veränderungen an der Skriptsprache vorgenommen werden oder die interpretierende Komponente (HTS-Parser) muss sich selbst um die Behandlung der Statusinformationen aus der Speicherstruktur kümmern. In beiden Fällen wäre dazu ein Eingriff in den Quellcode der jeweiligen Parser-Komponenten notwendig. In dieser Hinsicht ließe sich weiterhin die Austauschbarkeit der atomaren Skriptsprache ATS bzw. ihres Interpreters untersuchen.

Im Laufe der Arbeiten zur Automatisierung von Regressionstests ist die Idee entstanden, ein Verfahren zur automatisierten Generierung von Testskripten aus den Sequenzentabellen der Testdokumente (Web-Repository) zu entwerfen. Weitere Untersuchungen haben jedoch ergeben, dass ein solches Vorgehen nicht vorteilhaft wäre. In der Freiheit zur Formulierung von Aktions- und Auswertungsschritten wäre man sehr stark eingegrenzt. Für die Übersetzung eines Testschrittes in ein äquivalentes Skriptkommando, müsste er in seiner Struktur einer definierten Syntax entsprechen. Abweichungen bei der Formulierung wie „Schaltknopf drücken“ statt „Button anklicken“ wären nicht zulässig. Die Dokumente dienen der verbalen Beschreibung der durchzuführenden Arbeitsschritte, mit deren Hilfe ein Testfall auch manuell durchgeführt werden könnte. Eine starre Vorschrift zur Formulierung der Testschritte entspricht der Definition einer weiteren abstrakten Skriptsprache. Das Schreiben, Lesen und Verwalten von Sequenzentabellen in den Testdokumenten des Web-Repositorys würde für menschliche Benutzer damit nur unnötig erschwert werden.

Anhang A

Regel-Syntax der Komponente Matcher

Einleitung

Der *Matcher* (zu deutsch Vergleicher) gehört zu den Kernkomponenten, die in den Projekten *RCParser* und *ATOS* verwendet werden. Er wird bei allen Übersetzungs- und Umformungsvorgängen eingesetzt. Seine Funktionsweise ist mit der einer *Turing-Maschine* vergleichbar. In einem Zustand wird zu einer gegebenen Eingabe eine Regel gesucht und ggf. eine Ausgabe und ein Zustandswechsel erzeugt.

Der *Matcher* bezieht alle Regeln aus einer ASCII-Datei. Somit kann sein Verhalten auch nach der Kompilation angepasst, erweitert und beeinflusst werden. Die Regeln müssen in einer speziellen Form vorliegen, damit diese von der Komponente gelesen werden können. Die Form und Einschränkungen werden in diesem Dokument beschrieben.

Die Funktionsweise

Wie schon in der Einleitung beschrieben, funktioniert die *Matcher*-Komponente wie eine *Turing-Maschine*. Zu Beginn besitzt der *Matcher* einen Anfangszustand, der in der Regel-Datei festgelegt wird. Dann erhält er von dem Programm, das ihn verwendet eine Eingabe. Der *Matcher* versucht zum aktuellen Zustand und der Eingabe eine passende Regel zu finden.

Wird eine Regel gefunden, so wird diese interpretiert. Wenn es die Regel vorsieht, wird eine Ausgabe erzeugt, die an das Programm zurückgegeben wird. Zusätzlich kann die Regel dafür sorgen, daß sich der Zustand des *Matchers* ändert.

Ist keine passende Regel zu finden, wird dies dem benutzenden Komponente mitgeteilt und der *Matcher* bleibt in seinem aktuellen Zustand.

Dann beginnt der Vorgang wieder von vorne. Der *Matcher* wird primär dazu verwendet, syntaktische Umformungen oder Übersetzungen vorzunehmen. Aber auch als Datenbank kann er fungieren, wenn keine Ausgaben erzeugt werden sollen, sondern nur die syntaktische Struktur der Eingaben von Interesse ist. In diesem Falle bildet die Menge aller Regeln eine Datenbank zur Beschreibung des zulässigen Eingabeformates. In den Projekten *RCParser* und *ATOS* wird der Matcher für folgende Aufgaben verwendet:

- Übersetzung von Ressourcen-Dateien von Windows-Applikationen in das *URF*-Format (*RCParser*)
- Syntaktische und strukturelle Beschreibung der Sprache HTS
- Übersetzung von HTS-Kommandos in Skripte der Sprache ATS
- Einlesen von CTE-Diagrammen zur Generierung von Testskripten

Das Verhalten dieser Funktionen ist durch externe Regeldateien anpassbar, erweiterbar und beeinflussbar.

Die Syntax

Die Regeldatei

Die Regeln müssen ein Form einer ASCII-Datei vorliegen. Es ist sowohl das DOS- als auch das UNIX-Format zulässig. Somit lassen sich die Dateien mit einem normalen Texteditor erstellen.

Kommentare

Kommentare werden mit dem Zeichen `#` eröffnet und erstrecken sich dann bis zu dem Ende der Zeile. Einige Beispiele:

```
# Hier beginnt der Kommantar und geht bis hier
Das hier ist kein Kommantar # das hier hingegen schon
```

Die Zeile

Die gesamte Regelbeschreibung besteht aus Zeilen, die immer dem selben Aufbau folgen. Eine Zeile besteht aus durch , oder Leerzeichen getrennten Einträgen. Auch die Eingaben und Ausgaben des *Matcher*'s besitzen diese Form. Um das Prinzip der Zerlegung zu verstehen hilft ein Beispiel: Der Satz

```
Er kam, sagte "Hallo" und setzte sich.
```

wird zerlegt in

```
Er,kam,sagte,"Hallo",und,setzte,sich.
```

In einem solchen Format lassen sich Zeilen leichter verarbeiten. Die Trennzeichen (, und Leerzeichen) dürfen nur in Einträgen vorkommen, die von " umgeben sind.

Kopf der Regeldatei

Jede Regeldatei beginnt mit zwei Zeilen:

```
DESCRIPTION  "<Beschreibung>"
BEGINSTATE   "<Anfangszustand>"
```

Der Eintrag `<Beschreibung>` beinhaltet eine Beschreibung, die kurz die Funktion der Regeldatei beschreiben soll. Diese wird teilweise verwendet, um dem Benutzer die Auswahl einer Regeldatei zu erleichtern. Im Eintrag `<Anfangszustand>` steht eine Zeichenkette, die den Anfangszustand des *Matcher*'s bezeichnet. Der Kopf einer Regeldatei kann z.B. so aussehen:

```
DESCRIPTION  "Test-Regeln"
BEGINSTATE   "NORMAL"
```

Es können aber noch zusätzliche Informationen enthalten sein, die aber von einer Komponente stammen, die den *Matcher* benutzt.

Die Regel

Die Regeln bilden das Herzstück der Umformung. Sie beschreiben wann, welche Eingabe, welche Ausgabe erzeugt. Die Regeln entsprechen immer derselben Form:

```
RULE
 STATE    <Zustand>
 PATTERN  <Signatur>
 OUTPUT   <Ausgabe>
 (OUTPUT  <Ausgabe>)*
 NEWSTATE <Neuer Zustand>
ENDRULE
```

Wie zu erkennen ist, darf das Kommando `OUTPUT` mehrmals auftreten, so daß eine Ausgabe aus mehreren Zeilen bestehen kann. Es muß aber mind. einmal vorhanden sein. Eine genaue Beschreibung der einzelnen Kommandos folgt in den nächsten Abschnitten.

STATE

Die Anweisung `STATE` beschreibt, welchen Zustand der *Matcher* haben muß, damit die Regel angewendet werden kann. Die Anweisung hat die folgende Form: `STATE <Zustand>` Der in `Zustand` beschriebene Zustand kann aus bel. Kombinationen folgender Elemente bestehen:

Element	**Beschreibung**
`"<Zustand>"`	Beschreibt welchen Zustand der *Matcher* haben muß, damit die Regel angewendet werden darf. Die in `Zustand` beschriebene Zeichenkette muß mit der des aktuellen Zustands übereinstimmen. Dieses Element sollte maximal einmal vorhanden sein.
`<ZERO:<Variable>>`	Bei diesem Element darf die Regel nur angewendet werden, wenn der numerische Wert der mit `<Variable>` spezifizierten Variable dem Wert Null entspricht oder die Variable noch nicht existiert. Der Name der Variable `<Variable>` darf eine beliebige alphanumerische Zeichkette ohne Leerzeichen sein.
`<NOTZERO:<Variable>>`	Entspricht dem vorherigen Element außer daß hier der numerische Wert der angegebenen und vorhandenen Variable grösser als Null sein muß, damit die Regel angewendet werden darf.

Ein Beispiel für ein STATE-Kommando, kann folgendermaßen aussehen:

```
STATE "NORMAL",<ZERO:VAR1>,<NOTZERO:VAR2>
```

Eine Regel mit diesem Kommando darf nur angewendet werden, wenn der akt. Zustand NORMAL ist, die Variable VAR1 Null ist und die Variable VAR2 einen Wert grösser Null besitzt.

PATTERN

Die Anweisung PATTERN beschreibt, welche Form die Eingabe haben muß, damit die Regel anwendbar ist. Die Form der Anweisung sieht folgendermaßen aus:

```
PATTERN <Signatur>
```

Die in <Signatur> beschriebene Signatur kann eine Kombination der folgenden Elemente sein:

Element	Beschreibung
"<String>"	Damit die Regel passt, muß in dem selben Eintrag der Eingabe die in <String> beschriebene Zeichenkette vorhanden sein. Wenn man eine Zeichenkette beschreiben will, die von " umgeben ist, z.B. "Test", so muß das Element die Form "\"Test\"" haben.
<ANY>	Dieses Element setzt voraus, daß die Eingabe an der selben Stelle einen Eintrag hat. Der Inhalt kann aber beliebig sein.
<CONTINUE>	Dieses Element sollte am Ende der PATTERN-Anweisung stehen. Dieses Element entspricht dem <ANY>. Allerdings darf der selbe Eintrag in der Eingabe beliebig viele nachfolgende Einträge mit beliebigem Inhalt besitzen. Damit lassen sich auch Zeilen vergleichen, die eine variable Länge haben können.

<SAVE:<Variable>>	Speichert den Eintrag der Eingabe in die in <Variable> beschriebenen Variable. Somit lässt sich dieser dann für die Ausgabe verwenden. <Variable> darf eine beliebige alphanumerische Zeichenkette ohne Leerzeichen sein.
<CONTAINS:"<String>">	Damit die Regel angewendet werden kann, muß im selben Eintrag der Eingabezeile die in <String> beschriebene Zeichenkette an einer bel. Stelle enthalten sein. Auch hier muß das Zeichen " durch \" ausgedrückt werden.

Ein Beispiel für das PATTERN-Kommando ist

```
PATTERN "\"Hallo\"","sagte",<SAVE:WER>,<CONTINUE>
```

Auf folgende Eingabezeilen ließe sich die Regeln mit dieser PATTERN-Anweisung nicht anwenden:

```
"Auf Wiedersehen",sagte,Uwe,zu,Gerd
"Hallo",sagte,Gerd
Hallo,sagte,Uwe,laut
```

Aus folgenede Zeilen hingegen lässt sich die Regel anwenden:

```
"Hallo",sagte,Ralf,zu,Peter
"Hallo",sagte,Ralf,laut
```

Und in beiden Fällen würde der Name Ralf in der Variable WER gespeichert werden.

OUTPUT

In der OUTPUT-Anweisung wird die Ausgabe beschrieben, die erzeugt wird wenn die Regel angewendet werden konnte. Es können mehrere dieser Anweisungen aufeinander folgen, somit kann die Ausgabe auch aus mehreren Zeilen bestehen. Das Kommando OUTPUT hat folgende Form:

```
OUTPUT <Ausgabe>
```

In <Ausgabe> wird beschrieben, wie eine Ausgabezeile aussehen soll. Es ist die Kombination folgender Elemente erlaubt:

Element	Beschreibung
"<String>"	Hiermit wird direkt der Inhalt des Eintrages der Ausgabezeile bestimmt. <String> ist hierbei eine Zeichenkette. Um eine Zeichenkette zu beschreiben, die von " umschlossen wird (z.B. "Test") so muß das Element aussehen wie "\"Test\"".
<LOAD:<Variable>>	Lädt den Inhalt der zuvor in der Variable <Variable> gespeichert wurde und schreibt diesen in den aktuellen Eintrag der Ausgabezeile.
<NONE>	Erzeugt eine leere Ausgabezeile. Sollte nicht mit anderen Elementen zusammen verwendet werden.

Ein Beispiel wäre:

```
OUTPUT "\"Hallo\"","rief",<LOAD:WER>,"laut"
```

Hätte die Variable WER den Wert Peter, würde die Ausgabe

```
"Hallo",rief,Peter,laut
```

erzeugt werden, sofern die Regel angewendet werden konnte.

NEWSTATE

Beschreibt welchen Zustand der *Matcher* nach dem Anwenden einer Regel annehmen soll. Dabei lassen sich auch Werte von Variablen beeinflussen um diese später als zusätzliche Kriterien zum Zustand zu verwenden. Das NEWSTATE-Kommando hat folgende Form

```
NEWSTATE <Neuer Zustand>
```

wobei sich <Neuer Zustand> aus folgenden Elementen zusammensetzen darf:

Element	Beschreibung
"<Neuer Zustand>"	Beschreibt den direkten Zustand nach Anwenden der Regel. <Neuer Zustand> darf eine beliebige alphanumerische Zeichenkette sein. Diese wird nach dem Anwenden der Regel in den direkten Zustand übernommen. Dieses Element sollte nur einmal in einem NEWSTATE-Kommando vorkommen.

<INC:<Variable>>	Erhöht den numerischen Wert der Variable <Variable> um Eins. <Variable> darf eine beliebige alphanumerische Zeichenkette ohne Leerzeichen sein. Wenn die Variable zu diesem Zeitpunkt noch nicht existiert wird diese mit dem numerischen Wert Eins angelegt.
<DEC:<Variable>>	Ähnlich zu INC. Hier wird jedoch der Wert um Eins erniedrigt sofern der numerische Wert der Variable größer Null ist. Wenn die Variable noch nicht existiert, wird diese mit dem Wert Null angelegt.

Ein Beispiel für eine NEWSTATE-Anweisung ist:

```
NEWSTATE "Spezial",<INC:VAR>
```

Wenn die Regel mit dem Kommando angewendet werden konnte, hat der *Matcher* danach den Zustand Spezial und die Variable VAR wurde mit dem Wert Eins angelegt bzw. der numerische Wert um Eins erhöht.

Weitere Hinweise

Es sollte vermieden werden die selbe Variable für die Zustandsbeeinflussung (INC, DEC, ZERO und NOTZERO) und für die Pufferung von Einträgen (LOAD, SAVE) zu verwenden. Es ist zwar durchaus möglich, kann aber zu nicht vorhersehbaren Effekten führen, besonders wenn in den gespeicherten Einträgen Zeichenketten vorkommen. Empfehlenswert ist es auch einen Eintrag, der mit SAVE gesichert wurde in der selben Regel mit LOAD sofort zu verwenden. Ist das nicht der Fall, muß beachtet werden, dass damit nicht unfreiwillig die Daten von anderen Regeln überschrieben werden.

Die Pre-Parse Regel

Die Pre-Parse Regeln haben die Aufgabe, die Eingabezeilen vor dem eigentlichen Vergleichen umzuformen. Mit ihnen lassen sich mehrere Einträge einer Zeile zu einem Eintrag vereinen. Die Syntax einer solchen Regel hat die folgende Form:

```
PREPARSE <Signatur>
```

Die in <Signatur> spezifizierte Signatur kann sich aus folgenden Elementen zusammensetzen:

Element	Beschreibung
"<String>"	Wie bei PATTERN
<ANY>	Wie bei PATTERN
<CONCAT:"<Verbindung>">	An dieser Stelle werden alle überschüssigen Einträge zu einem zusammengefügt. Dabei werden sie mit der Zeichenkette in <Verbindung> verbunden. Um " zu verwenden, ist in <Verbindung> die Zeichenkette \" zu verwenden. Es sollten nur Einträge verbunden werden, die nicht von " umgeben sind. Sonst kann es zu Problemen kommen. Es ist auch zu beachten, das in dem resultierenden Eintrag Trennzeichen enthalten sein können auch wenn dieser nicht von " umgeben ist. Das ist davon abhängig, was in <Verbindung> beschrieben wurde.

Ein Beispiel: Mit der PREPARSE-Regel

```
PREPARSE "Peter","hat",<ANY>,<ANY>,<CONCAT:" und ">,<ANY>
```

und der Eingabezeile

```
Peter,hat,die,Staedte,Berlin,Moskau,Peking,Tokyo,besucht
```

wird die neue Eingabezeile

```
Peter,hat,die,Staedte,Berlin und Moskau und Peking und Tokyo,besucht
```

erzeugt. Das lässt sich z.B. verwenden um mehrere Eigenschaften in einen Eintrag zusammanzufassen um diesen später mit CONTAINS genauer zu untersuchen. Die PEPARSE-Regeln sollten zur Übersichtlichkeit zwischen den normalen Regeln und dem Kopf der Regeldatei stehen.

Beispiel einer Regeldatei

Es folgt ein Auszug einer im Projekt verwendeten Regeldatei. Diese wird für den *RCParser* verwendet, um Ressourcen-Dateien der Entwicklungsumgebung *Borland C++* in das „uniforme Ressourcen-Format“ (*URF*) zu übersetzen.

```
DESCRIPTION "Borland C/C++"
SYNTAX      ",|","//","/*","*/","\\"  # Syntaxregeln zum Parsen der Ressource
BEGINSTATE  "NORMAL"                  # Anfanszustand des Matchers

# PreParse-Regeln
#
PREPARSE,"CONTROL",<ANY>,<ANY>,<ANY>,<CONCAT:" ">,<ANY>,<ANY>,<ANY>,<ANY>
PREPARSE,"PUSHBUTTON",<ANY>,<ANY>,<ANY>,<ANY>,<ANY>,<ANY>,<CONCAT:" ">
PREPARSE,"DEFPUSHBUTTON",<ANY>,<ANY>,<ANY>,<ANY>,<ANY>,<ANY>,<CONCAT:" ">

# Regeln
#

# Dialogbloecke
#
RULE
   STATE    "NORMAL"
   PATTERN  <SAVE:ID>,"DIALOG",<CONTINUE>
   OUTPUT   <NONE>
   NEWSTATE "DIALOGBEGIN"
ENDRULE

RULE
   STATE    "DIALOGBEGIN"
   PATTERN  "CAPTION",<SAVE:CAPTION>
   OUTPUT   <NONE>
   NEWSTATE "DIALOGHEAD"
ENDRULE

RULE
   STATE    "DIALOGHEAD"
   PATTERN  "MENU",<SAVE:MENUID>
   OUTPUT   <NONE>
   NEWSTATE "DIALOGWITHMENU"
ENDRULE

RULE
   STATE    "DIALOGWITHMENU"
   PATTERN  "{"
   OUTPUT   "DIALOG",<LOAD:ID>,<LOAD:CAPTION>,<LOAD:MENUID>
   NEWSTATE "DIALOGBLOCK"
ENDRULE

RULE
   STATE    "DIALOGHEAD"
   PATTERN  "{"
   OUTPUT   "DIALOG",<LOAD:ID>,<LOAD:CAPTION>,"<NOMENU>"
   NEWSTATE "DIALOGBLOCK"
ENDRULE

RULE
   STATE    "DIALOGBLOCK"
   PATTERN  "}"
   OUTPUT   "ENDDIALOG"
   NEWSTATE "NORMAL"
ENDRULE
```

```
# Controls
#
RULE
   STATE    "DIALOGBLOCK"
   PATTERN  "CONTROL",<SAVE:CAPTION>,<SAVE:ID>,"\"BorBtn\"",<CONTINUE>
   OUTPUT   "CONTROL","BUTTON",<LOAD:ID>,<LOAD:CAPTION>
   NEWSTATE "DIALOGBLOCK"
ENDRULE

RULE
   STATE    "DIALOGBLOCK"
   PATTERN  "CONTROL",<SAVE:CAPTION>,<SAVE:ID>,"\"BUTTON\"",<CONTINUE>
   OUTPUT   "CONTROL","BUTTON",<LOAD:ID>,<LOAD:CAPTION>
   NEWSTATE "DIALOGBLOCK"
ENDRULE

RULE
   STATE    "DIALOGBLOCK"
   PATTERN  "PUSHBUTTON",<SAVE:CAPTION>,<SAVE:ID>,<CONTINUE>
   OUTPUT   "CONTROL","BUTTON",<LOAD:ID>,<LOAD:CAPTION>
   NEWSTATE "DIALOGBLOCK"
ENDRULE

# Menüs
#
RULE
   STATE    "NORMAL"
   PATTERN  <SAVE:ID>,"MENU"
   OUTPUT   <NONE>
   NEWSTATE "MENUBEGIN"
ENDRULE

RULE
   STATE    "NORMAL"
   PATTERN  <SAVE:ID>,"MENU",<CONTINUE>
   OUTPUT   <NONE>
   NEWSTATE "MENUBEGIN"
ENDRULE

RULE
   STATE    "MENUBEGIN"
   PATTERN  "{"
   OUTPUT   "MENU",<LOAD:ID>
   NEWSTATE "MENU"
ENDRULE

RULE
   STATE    "MENU",<ZERO:DEPTH>
   PATTERN  "}"
   OUTPUT   "ENDMENU"
   NEWSTATE "NORMAL"
ENDRULE

RULE
   STATE    "MENU"
   PATTERN  "MENUITEM","SEPARATOR"
   OUTPUT   <NONE>
   NEWSTATE "MENU"
ENDRULE
```

```
RULE
   STATE    "MENU"
   PATTERN  "MENUITEM",<SAVE:CAPTION>,<SAVE:ID>
   OUTPUT   "ITEM",<LOAD:ID>,<LOAD:CAPTION>
   NEWSTATE "MENU"
ENDRULE

RULE
   STATE    "MENU"
   PATTERN  "POPUP",<SAVE:CAPTION>
   OUTPUT   <NONE>
   NEWSTATE "POPUPBEGIN"
ENDRULE

RULE
   STATE    "POPUPBEGIN"
   PATTERN  "{"
   OUTPUT   "POPUP",<LOAD:CAPTION>
   NEWSTATE "MENU",<INC:DEPTH>
ENDRULE

RULE
   STATE    "MENU"
   PATTERN  "}"
   OUTPUT   "ENDPOPUP"
   NEWSTATE "MENU",<DEC:DEPTH>
ENDRULE
```

Anhang B

Beispiel Testfall Test_MJ.1

B.1 Testfall Test_MJ.1 als Sequenz im Web-Repository

Vorbereitung

Schritt	Aktionen	Erklärung
1.	Existenz aller Umgebungsdateien des XCTL-Systems im Programmverzeichnis überprüfen	Gültigen und startfähigen Ausgangszustand des XCTL-Systems sicherstellen
2.	i. Umbenennen der Datei `DEVELOP.INI` in `DEVELOP.BAK` ii. Kopieren der Datei `.\ini\TEST_DEVELOP.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `DEVELOP.INI`	Sicherung der originalen Datei `DEVELOP.INI` und Ersetzung durch eine präparierte Konfiguration
3.	i. Umbenennen der Datei `HARDWARE.INI` in `HARDWARE.BAK` ii. Kopieren der Datei `.\ini\OTOPO_HARDWARE.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `HARDWARE.INI`	Sicherung der originalen Datei `HARDWARE.INI` und Ersetzung durch eine präparierte Konfiguration

Testsequenz

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
1.	Starten des XCTL-Systems (Ausführen der `Develop.exe`)	Das Hauptfenster des XCTL-Systems öffnet sich
2.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	
3.	Hauptmenü: Ausführen -> Manuelle Justage...	Dialogbox "Manuelle Justage" öffnet sich
4.	"Aktueller Antrieb" auf **DF** setzen	Werte des Antriebs DF erscheinen in der Dialogbox
5.	i. "Neuer Winkel" auf **50.12** Grad setzen ii. **Enter** drücken	Antrieb DF setzt sich in Bewegung
6.	3 Sekunden warten (Motorbewegung abwarten)	"Winkel" steht auf 50.12 Sekunden
7.	Button **Relative Null** setzen anklicken	"Winkel" und "Neuer Winkel" stehen auf 0.00 Sekunden
8.	i. "Neuer Winkel" auf **-100.98** Grad setzen ii. **Enter** drücken	Antrieb DF setzt sich in Bewegung
9.	5 Sekunden warten (Motorbewegung abwarten)	"Winkel" steht auf -100.98 Sekunden
10.	Button **Relative Null** anklicken	i. Button "Relative Null aufheben" ist ausgegraut ii. "Winkel" und "Neuer Winkel" stehen auf -50.85 Sekunden
11.	i. "Neuer Winkel" auf **0** Grad setzen ii. **Enter** drücken	Antrieb DF setzt sich in Bewegung
12.	4 Sekunden warten (Motorbewegung abwarten)	"Winkel" steht auf 0.0 Sekunden
13.	Button **Verlassen** anklicken	Dialogbox "Manuelle Justage" schließt sich
14.	Hauptmenü: Datei -> Beenden	Das Hauptfenster des XCTL-Systems schließt sich
15.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	

Nachbereitung

Schritt	Aktionen	Erklärung
1.	Umbenennen der Datei `DEVELOP.BAK` in `DEVELOP.INI`	Wiederherstellung der originalen Datei `DEVELOP.INI`
2.	Umbenennen der Datei `HARDWARE.BAK` in `HARDWARE.INI`	Wiederherstellung der originalen Datei `HARDWARE.INI`

B.2 Testfall Test_MJ.1 als HTS-Sequenz

```
# Skriptdatei für den Testfall MJ.1
# Anwendungsfall Manuelle Justage
# 29.07.2002

########## Vorbereitung

EXISTS,TGT,"DEVELOP.INI"
EXISTS,TGT,"HARDWARE.INI"
EXISTS,TGT,"STANDARD.MAK"
EXISTS,TGT,"TESTDEV.DAT"

COPY,TGT,"DEVELOP.INI",TGT,"DEVELOP.BAK"
COPY,ENV,"TEST_DEVELOP.INI",TGT,"DEVELOP.INI"

COPY,TGT,"HARDWARE.INI",TGT,"HARDWARE.BAK"
COPY,ENV,"OTOPO_HARDWARE.INI",TGT,"HARDWARE.INI",FORCE

########## Testsequenz

START,""

WAIT,20000

ACTION,MAIN,MENU,CLICK,"Ausführen","Manuelle Justage..."

WINDOWEXISTS,"Manuelle Justage",YES

ACTION,"Manuelle Justage",COMBOBOX,SELECT,"DF","Aktueller Antrieb"
ACTION,"Manuelle Justage",EDITBOX,SET,"50.12","Neuer Winkel"

WAIT,3000

TEST,"Manuelle Justage",STATIC,NUM,"50.12","Winkel"

ACTION,"Manuelle Justage",BUTTON,CLICK,"Relative Null setzen"

TEST,"Manuelle Justage",BUTTON,ENABLED,"Relative Null aufheben"

TEST,"Manuelle Justage",STATIC,NUM,"0.0","Winkel"
TEST,"Manuelle Justage",EDITBOX,NUM,"0.0","Neuer Winkel"

ACTION,"Manuelle Justage",EDITBOX,SET,"-100.98","Neuer Winkel"

WAIT,5000

TEST,"Manuelle Justage",STATIC,NUM,"-100.98","Winkel"

ACTION,"Manuelle Justage",BUTTON,CLICK,"Relative Null aufheben"

TEST,"Manuelle Justage",BUTTON,DISABLED,"Relative Null aufheben"
TEST,"Manuelle Justage",STATIC,NUM,"-50.86",TOL,0.01,"Winkel"

ACTION,"Manuelle Justage",EDITBOX,SET,"0","Neuer Winkel"

WAIT,4000

TEST,"Manuelle Justage",STATIC,NUM,"0.00","Winkel"

ACTION,"Manuelle Justage",BUTTON,CLICK,"Verlassen"

WINDOWEXISTS,"Manuelle Justage",NO

ACTION,MAIN,MENU,CLICK,"Datei","Beenden"

WAIT,10000

########## Nachbereitung

CLEANUP

COPY,TGT,"DEVELOP.BAK",TGT,"DEVELOP.INI"
DELETE,TGT,"DEVELOP.BAK"

COPY,TGT,"HARDWARE.BAK",TGT,"HARDWARE.INI",FORCE
DELETE,TGT,"HARDWARE.BAK",FORCE
```

B.3 Testfall Test_MJ.1 als ATS-Sequenz

```
# HTS: EXISTS,TGT,"DEVELOP.INI"
FILEEXISTS,"G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.INI"


# HTS: EXISTS,TGT,"HARDWARE.INI"
FILEEXISTS,"G:\\Studium\\Diplom\\XCTL\\EXE\\HARDWARE.INI"


# HTS: EXISTS,TGT,"STANDARD.MAK"
FILEEXISTS,"G:\\Studium\\Diplom\\XCTL\\EXE\\STANDARD.MAK"


# HTS: EXISTS,TGT,"TESTDEV.DAT"
FILEEXISTS,"G:\\Studium\\Diplom\\XCTL\\EXE\\TESTDEV.DAT"


# HTS: COPY,TGT,"DEVELOP.INI",TGT,"DEVELOP.BAK"
FILECOPY,"G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.INI","G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.BAK"


# HTS: COPY,ENV,"TEST_DEVELOP.INI",TGT,"DEVELOP.INI"
FILECOPY,"G:\\Studium\\Diplom\\ATOS_XCTLProjekt\\ENV\\TEST_DEVELOP.INI","G:\\Studium\\Diplom\\XCTL\\...


# HTS: COPY,TGT,"HARDWARE.INI",TGT,"HARDWARE.BAK"
FILECOPY,"G:\\Studium\\Diplom\\XCTL\\EXE\\HARDWARE.INI","G:\\Studium\\Diplom\\XCTL\\EXE\\HARDWARE.BAK"


# HTS: COPY,ENV,"OTOPO_HARDWARE.INI",TGT,"HARDWARE.INI",FORCE
FILECOPY,"G:\\Studium\\Diplom\\ATOS_XCTLProjekt\\ENV\\OTOPO_HARDWARE.INI","G:\\Studium\\Diplom\\XCTL\\...


# HTS: START,""
LAUNCH,"G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.EXE ","Steuerprogramm"


# HTS: WAIT,20000
WAIT,20000


# HTS: ACTION,MAIN,MENU,CLICK,"Ausführen","Manuelle Justage..."
ISENABLED,MENU,MAIN,30011
POST,MAIN,NONE,273,30011,0
POST,MAIN,NONE,15,0,0


# HTS: WINDOWEXISTS,"Manuelle Justage",YES
IF
WAITFORWINDOW,"Manuelle Justage",500
ELSE
COMPARE,0,NUM,1
ENDIF


# HTS: ACTION,"Manuelle Justage",COMBOBOX,SELECT,"DF","Aktueller Antrieb"
ISENABLED,CONTROL,"Manuelle Justage",17000
SETFOCUS,"Manuelle Justage",17000
SEND,"Manuelle Justage",17000,333,0,"DF"
SEND,"Manuelle Justage",NONE,273,82536,0
WAIT,500
SEND,"Manuelle Justage",17000,347,0,0
SAVE,RESULT,TopItemIndex,NUM
SEND,"Manuelle Justage",17000,327,0,0
SAVE,RESULT,CurItemIndex,NUM
CALC,CurItemIndex,ADD,TopItemIndex
SEND,"Manuelle Justage",17000,328,CurItemIndex,""
COMPARE,LPARAM,STR,"DF"
```

```
# HTS: ACTION,"Manuelle Justage",EDITBOX,SET,"50.12","Neuer Winkel"
ISENABLED,CONTROL,"Manuelle Justage",11500
SETFOCUS,"Manuelle Justage",11500
SEND,"Manuelle Justage",11500,12,0,"50.12"
POST,"Manuelle Justage",11500,256,13,0


# HTS: WAIT,3000
WAIT,3000


# HTS: TEST,"Manuelle Justage",STATIC,NUM,"50.12","Winkel"
SEND,"Manuelle Justage",17700,13,260,""
COMPARE,LPARAM,NUM,"50.12"


# HTS: ACTION,"Manuelle Justage",BUTTON,CLICK,"Relative Null setzen"
ISENABLED,CONTROL,"Manuelle Justage",10800
SETFOCUS,"Manuelle Justage",10800
POST,"Manuelle Justage",10800,245,0,0
WAIT,500


# HTS: TEST,"Manuelle Justage",BUTTON,ENABLED,"Relative Null aufheben"
IF
ISENABLED,CONTROL,"Manuelle Justage",10500
ELSE
COMPARE,0,NUM,1
ENDIF


# HTS: TEST,"Manuelle Justage",STATIC,NUM,"0.0","Winkel"
SEND,"Manuelle Justage",17700,13,260,""
COMPARE,LPARAM,NUM,"0.0"


# HTS: TEST,"Manuelle Justage",EDITBOX,NUM,"0.0","Neuer Winkel"
SEND,"Manuelle Justage",11500,13,260,""
COMPARE,LPARAM,NUM,"0.0"


# HTS: ACTION,"Manuelle Justage",EDITBOX,SET,"-100.98","Neuer Winkel"
ISENABLED,CONTROL,"Manuelle Justage",11500
SETFOCUS,"Manuelle Justage",11500
SEND,"Manuelle Justage",11500,12,0,"-100.98"
POST,"Manuelle Justage",11500,256,13,0


# HTS: WAIT,5000
WAIT,5000


# HTS: TEST,"Manuelle Justage",STATIC,NUM,"-100.98","Winkel"
SEND,"Manuelle Justage",17700,13,260,""
COMPARE,LPARAM,NUM,"-100.98"


# HTS: ACTION,"Manuelle Justage",BUTTON,CLICK,"Relative Null aufheben"
ISENABLED,CONTROL,"Manuelle Justage",10500
SETFOCUS,"Manuelle Justage",10500
POST,"Manuelle Justage",10500,245,0,0
WAIT,500


# HTS: TEST,"Manuelle Justage",BUTTON,DISABLED,"Relative Null aufheben"
IFNOT
ISENABLED,CONTROL,"Manuelle Justage",10500
ELSE
COMPARE,0,NUM,1
ENDIF


# HTS: TEST,"Manuelle Justage",STATIC,NUM,"-50.86",TOL,0.01,"Winkel"
SEND,"Manuelle Justage",17700,13,260,""
COMPARE,LPARAM,NUM,"-50.86",TOL,0.01
```

```
# HTS: ACTION,"Manuelle Justage",EDITBOX,SET,"0","Neuer Winkel"
ISENABLED,CONTROL,"Manuelle Justage",11500
SETFOCUS,"Manuelle Justage",11500
SEND,"Manuelle Justage",11500,12,0,"0"
POST,"Manuelle Justage",11500,256,13,0


# HTS: WAIT,4000
WAIT,4000


# HTS: TEST,"Manuelle Justage",STATIC,NUM,"0.00","Winkel"
SEND,"Manuelle Justage",17700,13,260,""
COMPARE,LPARAM,NUM,"0.00"


# HTS: ACTION,"Manuelle Justage",BUTTON,CLICK,"Verlassen"
ISENABLED,CONTROL,"Manuelle Justage",3
SETFOCUS,"Manuelle Justage",3
POST,"Manuelle Justage",3,245,0,0
WAIT,500


# HTS: WINDOWEXISTS,"Manuelle Justage",NO
WAIT,500
IFNOT
WAITFORWINDOW,"Manuelle Justage",0
ELSE
COMPARE,0,NUM,1
ENDIF


# HTS: ACTION,MAIN,MENU,CLICK,"Datei","Beenden"
ISENABLED,MENU,MAIN,30049
POST,MAIN,NONE,273,30049,0
POST,MAIN,NONE,15,0,0


# HTS: WAIT,10000
WAIT,10000


# HTS: CLEANUP
CLEANUP


# HTS: COPY,TGT,"DEVELOP.BAK",TGT,"DEVELOP.INI"
FILECOPY,"G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.BAK","G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.INI"


# HTS: DELETE,TGT,"DEVELOP.BAK"
FILEDELETE,"G:\\Studium\\Diplom\\XCTL\\EXE\\DEVELOP.BAK"


# HTS: COPY,TGT,"HARDWARE.BAK",TGT,"HARDWARE.INI",FORCE
FILECOPY,"G:\\Studium\\Diplom\\XCTL\\EXE\\HARDWARE.BAK","G:\\Studium\\Diplom\\XCTL\\EXE\\HARDWARE.INI",FORCE


# HTS: DELETE,TGT,"HARDWARE.BAK",FORCE
FILEDELETE,"G:\\Studium\\Diplom\\XCTL\\EXE\\HARDWARE.BAK",FORCE
```

Anhang C

Konfigurationsdateien

C.1 Benutzer-Konfiguration

C.1.1 TEST_DEVELOP.INI

```
[Steuerprogramm]
Startup=Nothing
Environment=Expert
AutoCalibration=0
CreateIniDefaults=0
xo=30
yo=40
x1=935
y1=725

[Scan]
xo=22
yo=22
dx=600
dy=352

[AreaScan]
xo=247
yo=12
dx=628
dy=590

[AreaScanCCD]
xo=183
yo=77
dx=487
dy=373

[UserScal]
xo=116
yo=175

[Counter]
xo=9
yo=509
dx=266
dy=98
```

C.2 Hardware-Konfigurationen

Zur Erstellung der Dateien `topo_hardware.ini` und `diff_hardware.ini` als Ausgangspunkt für neue Hardware - Konfigurationen, dienten einige Dateien, die an den Arbeitsplätzen des physikalischen Instituts tatsächlich eingesetzt werden. Für den Teilbereich „Topographie“ wurden die folgenden Dateien herangezogen:

- `Rtk3.ini`
- `Rtk4wand.ini` (Grundlage für `topo_hardware.ini`)
- `Rtk7grau.ini`
- `steertk4.ini`

Für den Teilbereich „Diffraktometrie/Reflektometrie“ wurden die folgenden Dateien herangezogen:

- `devehrm1.ini`
- `devehrm2.ini` (Grundlage für `diff_hardware.ini`)
- `Hrm1.ini`
- `hrm2.ini`

Die Gegenüberstellung der Einträge, erfolgt nur für Antriebe (alle Abschnitte `[Motor<x>]`). In den folgenden Tabellen werden Differenzen zwischen den Werten in kursiver Schrift besonders hervorgehoben.

C.2.1 TOPO_HARDWARE.INI

Tabelle C.1: TOPO_HARDWARE.INI

Rtk3.ini	Rtk4wand.ini	Rtk7grau.ini	steerk4.ini	topo_hardware.ini
[Motor0]	[Motor0]	[Motor0]	[Motor0]	[Motor0]
Name=Beugung fein	Name=Beugung Fein	Name=Beugung fein	Name=Beugung Fein	Name=Beugung Fein
Unit=Sekunden	Unit=Sekunden	Unit=Sekunden	Unit=Sekunden	Unit=Sekunden
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
Digits=3	Digits=2	Digits=3	Digits=2	Digits=2
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
BoardId=1	BoardId=4	BoardId=1	BoardId=4	*BoardId=4*
MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=5000	MaxVelocity=8000	*MaxVelocity=8000*
Velocity=7999	Velocity=8000	Velocity=5000	Velocity=8000	*Velocity=8000*
SpeedScale=188.200	SpeedScale=188.200	SpeedScale=188.200	SpeedScale=188.200	SpeedScale=188.200
Torque=80	Torque=80	Torque=90	Torque=80	*Torque=80*
Gain=210	Gain=210	Gain=100	Gain=210	*Gain=210*
DynamicGain=110	DynamicGain=110	DynamicGain=100	DynamicGain=110	*DynamicGain=110*
Acceleration=7000	Acceleration=4000	Acceleration=2000	Acceleration=4000	*Acceleration=4000*
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
DistanceToZero=220038	DistanceToZero=213000	DistanceToZero=68762	DistanceToZero=213000	*DistanceToZero=213000*
MoveFirstToLimit=1	MoveFirstToZero=1	MoveFirstToLimit=1	MoveFirstToZero=1	
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
PositionMin=-215000	PositionMin=-215000	PositionMin=-220000	PositionMin=-215000	*PositionMin=-215000*
MinimalWidth=4	MinimalWidth=4	MinimalWidth=4	MinimalWidth=4	MinimalWidth=4
MaximalWidth=1000	MaximalWidth=51000	MaximalWidth=40000	MaximalWidth=51000	*MaximalWidth=51000*
PositionWidth=30	PositionWidth=20	PositionWidth=100	PositionWidth=20	*PositionWidth=20*
PositionMax=215000	PositionMax=215000	PositionMax=215000	PositionMax=215000	PositionMax=215000
AngleMin=-1255.600	AngleMin=-1255.60	AngleMin=-1183.600	AngleMin=-1255.60	*AngleMin=-1255.60*
AngleWidth=0.2000	AngleWidth=1.00000	AngleWidth=0.2000	AngleWidth=1.00000	*AngleWidth=1.00000*
AngleBias=0.000	AngleBias=0.00000	AngleBias=0.000	AngleBias=266.04703	*AngleBias=0.00000*
AngleMax=1255.600	AngleMax=1255.60	AngleMax=1156.700	AngleMax=1255.60	*AngleMax=1255.60*
Correction=1	Correction=1	Correction=1	Correction=1	Correction=1
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Hysteresis=50	Hysteresis=50	Hysteresis=50	Hysteresis=50	Hysteresis=50
DeltaPosition=7475	DeltaPosition=0	DeltaPosition=0	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=-2.1463e-16	Koeff_3=-2.540900e-16	Koeff_3=-2.1463e-16	Koeff_3=-2.540900e-16	*Koeff_3=-2.540900e-16*
Koeff_2=-1.53884e-09	Koeff_2=1.736300e-09	Koeff_2=-1.53884e-09	Koeff_2=1.736300e-09	*Koeff_2=1.736300e-09*
Koeff_1=5.38e-03	Koeff_1=5.840000e-03	Koeff_1=5.38e-03	Koeff_1=5.840000e-03	*Koeff_1=5.840000e-03*
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=0	Upwards=1	Upwards=1	Upwards=1	*Upwards=1*
	Position=0		Position=0	
	AngleZero=0.0		AngleZero=0.0	
RestartPosible=0	RestartPosible=1	RestartPosible=1	RestartPosible=1	*RestartPosible=1*
MaxFailure=4		MaxFailure=4		
DeccelerationPoint=20		DeccelerationPoint=20		

Rtk3.ini	Rtk4wand.ini	Rtk7grau.ini	steerk4.ini	topo_hardware.ini
[Motor1]	[Motor1]	[Motor1]	[Motor1]	[Motor1]
Name=Tilt	Name=TILT	Name=Tilt	Name=TILT	Name=TILT
Unit=Sekunden	Unit=Minuten	Unit=Sekunden	Unit=Minuten	*Unit=Minuten*
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
BoardId=2	BoardId=1	BoardId=2	BoardId=1	*BoardId=1*
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
Digits=2	Digits=1	Digits=3	Digits=1	*Digits=1*
MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=8000
Velocity=8000	Velocity=7999	Velocity=8000	Velocity=7999	*Velocity=7999*
SpeedScale=522.9	SpeedScale=522.9	SpeedScale=522.9	SpeedScale=522.9	SpeedScale=522.9
Torque=80	Torque=80	Torque=90	Torque=80	*Torque=80*
Gain=210	Gain=200	Gain=210	Gain=200	*Gain=200*
DynamicGain=110	DynamicGain=110	DynamicGain=110	DynamicGain=110	DynamicGain=110
Acceleration=7000	Acceleration=25000	Acceleration=7000	Acceleration=25000	*Acceleration=25000*
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
DistanceToZero=65915	DistanceToZero=82000	DistanceToZero=78192	DistanceToZero=82000	*DistanceToZero=82000*
MoveFirstToLimit=1	MoveFirstToZero=1	MoveFirstToLimit=1	MoveFirstToZero=1	
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
PositionMin=-84000	PositionMin=-84000	PositionMin=-82000	PositionMin=-84000	*PositionMin=-84000*
PositionWidth=400	PositionWidth=150	PositionWidth=400	PositionWidth=150	*PositionWidth=150*
PositionMax=84000	PositionMax=77000	PositionMax=70000	PositionMax=77000	*PositionMax=77000*
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=10000	MaximalWidth=800	MaximalWidth=10000	MaximalWidth=800	*MaximalWidth=800*
AngleMin=-18990.38	AngleMin=-314.3	AngleMin=-18990.384	AngleMin=-314.3	*AngleMin=-314.3*
AngleWidth=10.000	AngleWidth=0.30000	AngleWidth=0.5000	AngleWidth=0.30000	*AngleWidth=0.30000*
AngleBias=-5148.70	AngleBias=18.86052	AngleBias=0.000	AngleBias=-58.69963	*AngleBias=18.86052*
AngleMax=16211.30	AngleMax=288.1	AngleMax=16211.303	AngleMax=288.1	*AngleMax=288.1*
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Correction=1	Correction=1	Correction=1	Correction=1	Correction=1
Direction=0	Direction=0	Direction=0	Direction=0	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=23254	DeltaPosition=0	DeltaPosition=0	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=-2.67430e-13	Koeff_3=-2.67430e-13	Koeff_3=2.0126803e-13	Koeff_3=-2.67430e-13	Koeff_3=-2.67430e-13
Koeff_2=4.910500e-08	Koeff_2=4.910500e-08	Koeff_2=5.524973e-08	Koeff_2=4.910500e-08	Koeff_2=4.910500e-08
Koeff_1=2.245300e-01	Koeff_1=2.245300e-01	Koeff_1=2.3159005e-01	Koeff_1=2.245300e-01	Koeff_1=2.245300e-01
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1
	Position=0		Position=0	
	AngleZero=0.0		AngleZero=0.0	
RestartPosible=0	RestartPosible=1	RestartPosible=1	RestartPosible=1	*RestartPosible=1*
MaxFailure=3		MaxFailure=3		
DeccelerationPoint=20		DeccelerationPoint=20		

Rtk3.ini	Rtk4wand.ini	Rtk7grau.ini	steerk4.ini	topo_hardware.ini
[Motor2]	[Motor2]	[Motor2]	[Motor2]	[Motor2]
Name=Beugung grob	Name=Beugung Grob	Name=Beugung grob	Name=Beugung Grob	Name=Beugung Grob
Unit=Sekunden	Unit=Sekunden	Unit=Sekunden	Unit=Sekunden	Unit=Sekunden
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
BoardId=3	BoardId=2	BoardId=3	BoardId=2	*BoardId=2*
MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=5000	MaxVelocity=8000	*MaxVelocity=8000*
Velocity=8000	Velocity=5879	Velocity=5000	Velocity=5879	*Velocity=5879*
SpeedScale=5.882	SpeedScale=5.882	SpeedScale=5.882	SpeedScale=5.882	SpeedScale=5.882
Torque=100	Torque=80	Torque=100	Torque=80	Torque=80
Gain=220	Gain=200	Gain=40	Gain=200	*Gain=200*
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
Digits=1	Digits=1	Digits=3	Digits=1	*Digits=1*
DynamicGain=110	DynamicGain=110	DynamicGain=30	DynamicGain=110	*DynamicGain=110*
Acceleration=7000	Acceleration=25000	Acceleration=200	Acceleration=25000	*Acceleration=25000*
DeccelerationPoint=20		DeccelerationPoint=20		
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
DistanceToZero=85941	DistanceToZero=74000	DistanceToZero=85000	DistanceToZero=74000	*DistanceToZero=74000*
MoveFirstToLimit=1	MoveFirstToZero=1	MoveFirstToLimit=1	MoveFirstToZero=1	
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
PositionMin=-76000	PositionMin=-76000	PositionMin=-95000	PositionMin=-76000	*PositionMin=-76000*
PositionWidth=750	PositionWidth=750	PositionWidth=50	PositionWidth=750	*PositionWidth=750*
PositionMax=74000	PositionMax=74000	PositionMax=84000	PositionMax=74000	*PositionMax=74000*
AngleMin=-15399.4	AngleMin=-15399.4	AngleMin=-17305.680	AngleMin=-15399.4	*AngleMin=-15399.4*
AngleWidth=1.00	AngleWidth=10.00000	AngleWidth=10.0000	AngleWidth=10.00000	*AngleWidth=10.00000*
AngleBias=0.0	AngleBias=0.00000	AngleBias=0.000	AngleBias=0.00000	AngleBias=0.00000
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=800	MaximalWidth=800	MaximalWidth=10000	MaximalWidth=800	*MaximalWidth=800*
AngleMax=15815.6	AngleMax=15815.6	AngleMax=19571.900	AngleMax=15815.6	*AngleMax=15815.6*
Correction=1	Correction=1	Correction=1	Correction=1	Correction=1
Orientation=0	Orientation=0	Orientation=0	Orientation=0	
Direction=1	Direction=1	Direction=0	Direction=1	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=-6513	DeltaPosition=0	DeltaPosition=0	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=-3.191500e-14	Koeff_3=-3.191500e-14	Koeff_3=3.81359e-14	Koeff_3=-3.191500e-14	*Koeff_3=-3.191500e-14*
Koeff_2=-1.483100e-09	Koeff_2=-1.483100e-09	Koeff_2=-1.35782e-09	Koeff_2=-1.483100e-09	*Koeff_2=-1.483100e-09*
Koeff_1=-2.081000e-01	Koeff_1=-2.081000e-01	Koeff_1=-2.0602e-01	Koeff_1=-2.081000e-01	*Koeff_1=-2.081000e-01*
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1
	Position=0		Position=0	
	AngleZero=0.0		AngleZero=0.0	
RestartPosible=0	RestartPosible=1	RestartPosible=0	RestartPosible=1	*RestartPosible=1*
MaxFailure=4		MaxFailure=3		

Rtk3.ini	Rtk4wand.ini	Rtk7grau.ini	steerk4.ini	topo_hardware.ini
[Motor3]	[Motor3]	[Motor3]	[Motor3]	[Motor3]
Name=Azimutal Rot	Name=Azimutal Rot	Name=Rotation	Name=Azimutal Rot	*Name=Azimutal Rot*
Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad
Type=C-832	Type=C-832	Type=C-832	Type=C-832	Type=C-832
IOAddr=0x210	IOAddr=0x234	IOAddrX=0x210	IOAddr=0x234	*IOAddr=0x234*
BoardId=0	BoardId=0	BoardId=0	BoardId=0	BoardId=0
MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=8000	MaxVelocity=8000
Velocity=8000	Velocity=5881	Velocity=8000	Velocity=5881	*Velocity=5881*
SpeedScale=2000.0	SpeedScale=5.882	SpeedScale=2000.0	SpeedScale=5.882	*SpeedScale=5.882*
Digits=2	Digits=2	Digits=2	Digits=2	Digits=2
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
Gain=1200	Gain=400	Gain=1200	Gain=400	*Gain=400*
DynamicGain=10	DynamicGain=10	DynamicGain=10	DynamicGain=10	DynamicGain=10
IntegralLimit=300	IntegralLimit=300	IntegralLimit=300	IntegralLimit=300	IntegralLimit=300
IntegralGain=10	IntegralGain=10	IntegralGain=10	IntegralGain=10	IntegralGain=10
Acceleration=1000	Acceleration=50	Acceleration=1000	Acceleration=50	*Acceleration=50*
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
DistanceToZero=0	DistanceToZero=0	DistanceToZero=0	DistanceToZero=0	DistanceToZero=0
MoveFirstToLimt=1	MoveFirstToZero=1	MoveFirstToLimt=1	MoveFirstToZero=1	
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
InitialMove=0	InitialMove=0	InitialMove=0	InitialMove=0	InitialMove=0
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
PositionMin=-676000	PositionMin=-676000	PositionMin=-676000	PositionMin=-676000	PositionMin=-676000
PositionWidth=5000	PositionWidth=1868	PositionWidth=5000	PositionWidth=1868	*PositionWidth=1868*
PositionMax=674000	PositionMax=674000	PositionMax=674000	PositionMax=674000	PositionMax=674000
AngleMin=-296.91	AngleMin=-296.91	AngleMin=-296.91	AngleMin=-296.91	AngleMin=-296.91
AngleWidth=0.100	AngleWidth=0.10000	AngleWidth=0.100	AngleWidth=0.10000	AngleWidth=0.10000
AngleBias=-253.70	AngleBias=-44.45327	AngleBias=0.00	AngleBias=-51.79048	*AngleBias=-44.45327*
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=800	MaximalWidth=800	MaximalWidth=800	MaximalWidth=800	MaximalWidth=800
AngleMax=296.04	AngleMax=296.04	AngleMax=296.04	AngleMax=296.04	AngleMax=296.04
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=583085	DeltaPosition=0	DeltaPosition=0	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=1.581201e+00	Koeff_1=1.581201e+00	Koeff_1=1.581201e+00	Koeff_1=1.581201e+00	Koeff_1=1.581201e+00
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1
	AngleZero=0.0		AngleZero=0.0	
RestartPosible=1	RestartPosible=1	RestartPosible=1	RestartPosible=1	RestartPosible=1
MaxFailure=3		MaxFailure=3		

Rtk3.ini	Rtk4wand.ini	Rtk7grau.ini	steerk4.ini	topo_hardware.ini
[Motor5]	[Motor4]	[Motor4]	[Motor4]	*[Motor4]*
Name=Kollimatoralt	Name=Kollimator	Name=Kollimator	Name=Kollimator	*Name=Kollimator*
Unit=Mikrometer	Unit=Mikrometer	Unit=Mikrometer	Unit=Mikrometer	Unit=Mikrometer
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
BoardId=4	BoardId=3	BoardId=4	BoardId=3	*BoardId=3*
MaxVelocity=1000	MaxVelocity=800	MaxVelocity=1000	MaxVelocity=800	*MaxVelocity=800*
Velocity=1000	Velocity=799	Velocity=1000	Velocity=799	*Velocity=799*
SpeedScale=29.63	SpeedScale=29.63	SpeedScale=29.63	SpeedScale=29.63	SpeedScale=29.63
Torque=120	Torque=80	Torque=120	Torque=80	*Torque=80*
Gain=70	Gain=180	Gain=70	Gain=180	*Gain=180*
DynamicGain=120	DynamicGain=90	DynamicGain=120	DynamicGain=90	*DynamicGain=90*
Digits=1	Digits=1	Digits=1	Digits=1	Digits=1
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
Acceleration=1000	Acceleration=2500	Acceleration=1000	Acceleration=2500	*Acceleration=2500*
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
DistanceToZero=13801	DistanceToZero=13614	DistanceToZero=13801	DistanceToZero=13614	*DistanceToZero=13614*
MoveFirstToLimit=1	MoveFirstToZero=1	MoveFirstToLimit=1	MoveFirstToZero=1	
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
InitialMove=1	InitialMove=0	InitialMove=1	InitialMove=0	InitialMove=0
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
PositionMin=-18000	PositionMin=-18500	PositionMin=-18000	PositionMin=-18500	*PositionMin=-18500*
PositionWidth=10	PositionWidth=150	PositionWidth=10	PositionWidth=150	*PositionWidth=150*
PositionMax=18000	PositionMax=19000	PositionMax=18000	PositionMax=19000	*PositionMax=19000*
AngleMin=-1063.8	AngleMin=-1093.3	AngleMin=-1063.8	AngleMin=-1093.3	*AngleMin=-1093.3*
AngleWidth=1.00	AngleWidth=0.50000	AngleWidth=1.00	AngleWidth=0.50000	*AngleWidth=0.50000*
AngleBias=0.0	AngleBias=0.00000	AngleBias=0.0	AngleBias=64.71450	*AngleBias=0.00000*
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=500	MaximalWidth=300	MaximalWidth=500	MaximalWidth=300	MaximalWidth=300
AngleMax=1063.8	AngleMax=1122.9	AngleMax=1063.8	AngleMax=1122.9	AngleMax=1122.9
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=0	Direction=0	Direction=0	Direction=0	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=21	DeltaPosition=0	DeltaPosition=0	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=5.91e-2	Koeff_1=5.91e-2	Koeff_1=5.91e-2	Koeff_1=5.91e-2	Koeff_1=5.91e-2
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=0	*Upwards=1*
	Position=850		Position=850	
	AngleZero=0.0		AngleZero=0.0	
RestartPosible=0	RestartPosible=1	RestartPosible=0	RestartPosible=1	*RestartPosible=1*
MaxFailure=3		MaxFailure=3		
IntegralGain=50				
IntegralLimit=1000				

Rtk3.ini	Rtk4wand.ini	Rtk7grau.ini	steerk4.ini	topo_hardware.ini
[Motor4]		[Motor5]		*[Motor5]*
Name=Kollimator		Name=Kollimator_neu		*Name=Kollimator_neu*
Unit=Mikrometer		Unit=Millimeter		Unit=Millimeter
Type=C-832		Type=C-832		Type=C-832
IOAddrX=0x210		IOAddrX=0x210		IOAddrX=0x210
BoardId=1		BoardId=1		BoardId=1
MaxVelocity=2000		MaxVelocity=20000		MaxVelocity=20000
Velocity=2000		Velocity=20000		Velocity=20000
SpeedScale=29.63		SpeedScale=29.63		SpeedScale=29.63
Torque=180		Torque=180		Torque=180
Gain=2800		Gain=2800		Gain=2800
DynamicGain=0		DynamicGain=0		DynamicGain=0
Digits=1		Digits=1		Digits=1
DeathBand=1		DeathBand=1		DeathBand=1
Acceleration=80		Acceleration=80		Acceleration=80
DeccelerationPoint=20		DeccelerationPoint=20		DeccelerationPoint=20
RemoveLimit=4000		RemoveLimit=4000		RemoveLimit=4000
InitialMove=1		InitialMove=1		InitialMove=1
InitialAngle=0.0		InitialAngle=0.0		InitialAngle=0.0
IndexLine=1		IndexLine=1		IndexLine=1
MoveFirstToLimit=0		MoveFirstToLimit=0		MoveFirstToLimit=0
DistanceToZero=14237		DistanceToZero=16506		*DistanceToZero=16506*
PositionMin=-40000		PositionMin=-16950.		*PositionMin=-16950.*
PositionWidth=20		PositionWidth=20		PositionWidth=20
PositionMax=21995		PositionMax=16950		*PositionMax=16950*
AngleMin=-2000.0		AngleMin=-1000.0		*AngleMin=-1000.0*
AngleWidth=1.20		AngleWidth=5.42		*AngleWidth=5.42*
AngleBias=0.0		AngleBias=0.0		AngleBias=0.0
MinimalWidth=2		MinimalWidth=2		MinimalWidth=2
MaximalWidth=500		MaximalWidth=500		MaximalWidth=500
AngleMax=1000.0		AngleMax=1000.0		AngleMax=1000.0
Orientation=1		Orientation=1		
Direction=0		Direction=0		
Hysteresis=0		Hysteresis=0		Hysteresis=0
MaxFailure=0		MaxFailure=0		MaxFailure=0
DeltaPosition=-5255		DeltaPosition=0		*DeltaPosition=0*
Correction=0		Correction=0		Correction=0
Koeff_3=0.0		Koeff_3=0.0		Koeff_3=0.0
Koeff_2=0.0		Koeff_2=0.0		Koeff_2=0.0
Koeff_1=5.91e-2		Koeff_1=5.91e-2		Koeff_1=5.91e-2
Koeff_0=0.0		Koeff_0=0.0		
Upwards=0		Upwards=1		*Upwards=1*
RestartPosible=0		RestartPosible=1		*RestartPosible=1*
IntegralGain=50		IntegralGain=50		IntegralGain=50
IntegralLimit=1000		IntegralLimit=1000		IntegralLimit=1000

C.2.2 DIFF_HARDWARE.INI

Tabelle C.2: DIFF_HARDWARE.INI

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[Motor0]	[Motor0]	[Motor0]	[Motor0]	[Motor0]
Name=Theta	Name=Theta	Name=Theta	Name=Theta	Name=Theta
Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad
Type=C-832	Type=C-832	Type=C-832	Type=C-832	Type=C-832
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
Digits=3	Digits=3	Digits=3	Digits=3	Digits=3
IOAddr=0x210	IOAddr=0x210	IOAddr=0x210	IOAddr=0x210	IOAddr=0x210
BoardId=0	BoardId=0	BoardId=0	BoardId=0	BoardId=0
MaxVelocity=128000	MaxVelocity=100000	MaxVelocity=128000	MaxVelocity=100000	*MaxVelocity=100000*
Velocity=127997	Velocity=100000	Velocity=127998	Velocity=100000	*Velocity=100000*
SpeedScale=17000.0	SpeedScale=17000.0	SpeedScale=17000.0	SpeedScale=17000.0	SpeedScale=17000.0
Gain=900	Gain=600	Gain=900	Gain=600	*Gain=600*
DynamicGain=90	DynamicGain=60	DynamicGain=90	DynamicGain=60	*DynamicGain=60*
IntegralGain=5	IntegralGain=5	IntegralGain=5	IntegralGain=5	IntegralGain=5
IntegralLimit=4000	IntegralLimit=4000	IntegralLimit=4000	IntegralLimit=4000	IntegralLimit=4000
Acceleration=20	Acceleration=10	Acceleration=20	Acceleration=10	*Acceleration=10*
RemoveLimit=14400	RemoveLimit=14400	RemoveLimit=14400	RemoveLimit=14400	RemoveLimit=14400
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1
IndexLine=1	IndexLine=1	IndexLine=1	IndexLine=1	IndexLine=1
DistanceToZero=1076248	DistanceToZero=1116328	DistanceToZero=1076248	DistanceToZero=1116328	*DistanceToZero=1116328*
PositionMin=-1400000	PositionMin=-1400000	PositionMin=-1400000	PositionMin=-1400000	PositionMin=-1400000
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=20000	MaximalWidth=20000	MaximalWidth=20000	MaximalWidth=20000	MaximalWidth=20000
PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20
PositionMax=200000	PositionMax=200000	PositionMax=200000	PositionMax=200000	PositionMax=200000
AngleMin=-13.889	AngleMin=-13.889	AngleMin=-13.889	AngleMin=-13.889	AngleMin=-13.889
AngleWidth=0.5000	AngleWidth=0.0050	AngleWidth=0.5000	AngleWidth=0.0050	*AngleWidth=0.0050*
AngleBias=0.000	AngleBias=0.000	AngleBias=0.000	AngleBias=0.000	AngleBias=0.000
AngleMax=97.222	AngleMax=97.222	AngleMax=97.222	AngleMax=97.222	AngleMax=97.222
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=0	Direction=0	Direction=0	Direction=0	
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=72000	DeltaPosition=-705600	DeltaPosition=0	DeltaPosition=-705600	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=-2.500e-1	Koeff_1=-2.500e-1	Koeff_1=-2.500e-1	Koeff_1=-2.500e-1	Koeff_1=-2.500e-1
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=0	*Upwards=1*
AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	
MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0
RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=1

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[Motor1]	[Motor1]	[Motor1]	[Motor1]	[Motor1]
Name=Omega	Name=Omega	Name=Omega	Name=Omega	Name=Omega
Type=C-832	Type=C-832	Type=C-832	Type=C-832	Type=C-832
Digits=4	Digits=4	Digits=4	Digits=4	Digits=4
Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad
IOAddr=0x210	IOAddr=0x210	IOAddr=0x210	IOAddr=0x210	IOAddr=0x210
BoardId=1	BoardId=1	BoardId=1	BoardId=1	BoardId=1
MaxVelocity=120000	MaxVelocity=50000	MaxVelocity=120000	MaxVelocity=50000	*MaxVelocity=50000*
Velocity=119998	Velocity=50000	Velocity=119999	Velocity=50000	*Velocity=50000*
DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1	DeathBand=1
SpeedScale=17000.0	SpeedScale=17000.0	SpeedScale=17000.0	SpeedScale=17000.0	SpeedScale=17000.0
Gain=400	Gain=600	Gain=400	Gain=600	*Gain=600*
DynamicGain=60	DynamicGain=60	DynamicGain=60	DynamicGain=60	DynamicGain=60
IntegralGain=5	IntegralGain=5	IntegralGain=5	IntegralGain=5	IntegralGain=5
IntegralLimit=4000	IntegralLimit=4000	IntegralLimit=4000	IntegralLimit=4000	IntegralLimit=4000
Acceleration=20	Acceleration=10	Acceleration=20	Acceleration=10	*Acceleration=10*
RemoveLimit=14400	RemoveLimit=14400	RemoveLimit=14400	RemoveLimit=14400	RemoveLimit=14400
DistanceToZero=886242	DistanceToZero=877941	DistanceToZero=886242	DistanceToZero=877941	*DistanceToZero=877941*
MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=1	IndexLine=1	IndexLine=1	IndexLine=1	IndexLine=1
PositionMin=-1400000	PositionMin=-1400000	PositionMin=-1400000	PositionMin=-1400000	PositionMin=-1400000
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=20000	MaximalWidth=20000	MaximalWidth=20000	MaximalWidth=20000	MaximalWidth=20000
PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20
PositionMax=150000	PositionMax=150000	PositionMax=150000	PositionMax=150000	PositionMax=150000
AngleMin=-10.4167	AngleMin=-10.4167	AngleMin=-10.4167	AngleMin=-10.4167	AngleMin=-10.4167
AngleWidth=0.00100	AngleWidth=0.05000	AngleWidth=0.00100	AngleWidth=0.05000	*AngleWidth=0.05000*
AngleBias=0.0000	AngleBias=0.0000	AngleBias=0.0000	AngleBias=0.0000	AngleBias=0.0000
AngleMax=97.2222	AngleMax=97.2222	AngleMax=97.2222	AngleMax=97.2222	AngleMax=97.2222
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=-288000	DeltaPosition=-359983	DeltaPosition=0	DeltaPosition=-359983	*DeltaPosition=0*
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=-2.500e-1	Koeff_1=-2.500e-1	Koeff_1=-2.500e-1	Koeff_1=-2.500e-1	Koeff_1=-2.500e-1
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1
AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	
MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0
RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=1

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[Motor2]	[Motor2]	[Motor2]	[Motor2]	[Motor2]
Name=Psi	Name=Psi	Name=Psi	Name=Psi	Name=Psi
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad
BoardId=1	BoardId=1	BoardId=1	BoardId=1	BoardId=1
MaxVelocity=3200	MaxVelocity=3200	MaxVelocity=3200	MaxVelocity=3200	MaxVelocity=3200
DeathBand=2	DeathBand=8	DeathBand=2	DeathBand=8	DeathBand=2
Digits=2	Digits=2	Digits=2	Digits=2	Digits=2
Velocity=3200	Velocity=3200	Velocity=3200	Velocity=3200	Velocity=3200
SpeedScale=3200.0	SpeedScale=3200.0	SpeedScale=3200.0	SpeedScale=3200.0	SpeedScale=3200.0
Torque=120	Torque=120	Torque=120	Torque=120	Torque=120
Gain=200	Gain=200	Gain=200	Gain=200	Gain=200
DynamicGain=150	DynamicGain=150	DynamicGain=150	DynamicGain=150	DynamicGain=150
Acceleration=5800	Acceleration=3000	Acceleration=5800	Acceleration=3000	*Acceleration=3000*
DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
DistanceToZero=35344	DistanceToZero=35344	DistanceToZero=35344	DistanceToZero=35344	DistanceToZero=35344
RemoveLimit=2000	RemoveLimit=2000	RemoveLimit=2000	RemoveLimit=2000	RemoveLimit=2000
PositionMin=-20480	PositionMin=-20480	PositionMin=-20480	PositionMin=-20480	PositionMin=-20480
MinimalWidth=4	MinimalWidth=4	MinimalWidth=4	MinimalWidth=4	MinimalWidth=4
MaximalWidth=512	MaximalWidth=512	MaximalWidth=512	MaximalWidth=512	MaximalWidth=512
PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20
PositionMax=20480	PositionMax=20480	PositionMax=20480	PositionMax=20480	PositionMax=20480
AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00
AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100
AngleBias=0.00	AngleBias=0.00	AngleBias=0.00	AngleBias=0.00	AngleBias=0.00
AngleMax=10.00	AngleMax=10.00	AngleMax=10.00	AngleMax=10.00	AngleMax=10.00
DeltaPosition=-2856	DeltaPosition=929	DeltaPosition=-18	DeltaPosition=752	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=1.7578125	Koeff_1=8.75	Koeff_1=1.7578125	Koeff_1=8.75	*Koeff_1=8.75*
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0
RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=1
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
Upwards=1	Upwards=1	Upwards=1	Upwards=0	*Upwards=1*

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[Motor3]	[Motor3]	[Motor3]	[Motor3]	[Motor3]
Name=Phi	Name=Phi	Name=Phi	Name=Phi	Name=Phi
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad	Unit=Grad
BoardId=2	BoardId=2	BoardId=2	BoardId=2	BoardId=2
DeathBand=2	DeathBand=8	DeathBand=2	DeathBand=8	*DeathBand=8*
Digits=2	Digits=2	Digits=2	Digits=2	Digits=2
MaxVelocity=2200	MaxVelocity=2200	MaxVelocity=2200	MaxVelocity=2200	MaxVelocity=2200
Velocity=2200	Velocity=2200	Velocity=2200	Velocity=2200	Velocity=2200
SpeedScale=3200.0	SpeedScale=3200.0	SpeedScale=3200.0	SpeedScale=3200.0	SpeedScale=3200.0
Torque=126	Torque=120	Torque=126	Torque=120	*Torque=120*
Gain=200	Gain=200	Gain=200	Gain=200	Gain=200
DynamicGain=100	DynamicGain=150	DynamicGain=100	DynamicGain=150	*DynamicGain=150*
Acceleration=5600	Acceleration=3000	Acceleration=5600	Acceleration=3000	*Acceleration=3000*
DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
DistanceToZero=33325	DistanceToZero=33325	DistanceToZero=33325	DistanceToZero=33325	DistanceToZero=33325
RemoveLimit=2000	RemoveLimit=2000	RemoveLimit=2000	RemoveLimit=2000	RemoveLimit=2000
PositionMin=-20480	PositionMin=-20480	PositionMin=-20480	PositionMin=-20480	PositionMin=-20480
MinimalWidth=5	MinimalWidth=5	MinimalWidth=5	MinimalWidth=5	MinimalWidth=5
MaximalWidth=512	MaximalWidth=512	MaximalWidth=512	MaximalWidth=512	MaximalWidth=512
PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20	PositionWidth=20
PositionMax=20480	PositionMax=20480	PositionMax=20480	PositionMax=20480	PositionMax=20480
AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00
AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100
AngleBias=0.00	AngleBias=0.00	AngleBias=0.00	AngleBias=0.00	AngleBias=0.00
AngleMax=10.00	AngleMax=10.00	AngleMax=10.00	AngleMax=10.00	AngleMax=10.00
DeltaPosition=-332	DeltaPosition=674	DeltaPosition=0	DeltaPosition=674	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=1.7578125	Koeff_1=8.75	Koeff_1=1.7578125	Koeff_1=8.75	*Koeff_1=1.7578125*
Koeff_0=1e-7	Koeff_0=1e-7	Koeff_0=1e-7	Koeff_0=1e-7	
RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=1
MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[Motor4]	[Motor4]	[Motor4]	[Motor4]	[Motor4]
Name=Y	Name=Y	Name=Y	Name=Y	Name=Y
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
Unit=Millimeter	Unit=Millimeter	Unit=Millimeter	Unit=Millimeter	Unit=Millimeter
BoardId=4	BoardId=4	BoardId=4	BoardId=4	BoardId=4
MaxVelocity=3500	MaxVelocity=3500	MaxVelocity=3500	MaxVelocity=3500	MaxVelocity=3500
DeathBand=5	DeathBand=9	DeathBand=5	DeathBand=9	DeathBand=5
Digits=3	Digits=3	Digits=3	Digits=3	Digits=3
Velocity=3497	Velocity=3499	Velocity=3497	Velocity=3498	Velocity=3497
SpeedScale=10000.0	SpeedScale=10000.0	SpeedScale=10000.0	SpeedScale=10000.0	SpeedScale=10000.0
Torque=120	Torque=120	Torque=120	Torque=120	Torque=120
Gain=200	Gain=200	Gain=200	Gain=200	Gain=200
DynamicGain=150	DynamicGain=150	DynamicGain=150	DynamicGain=150	DynamicGain=150
Acceleration=3000	Acceleration=3000	Acceleration=3000	Acceleration=3000	Acceleration=3000
DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1	MoveFirstToLimit=1
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
DistanceToZero=31722	DistanceToZero=31722	DistanceToZero=31722	DistanceToZero=31722	DistanceToZero=31722
PositionMin=-28672	PositionMin=-28672	PositionMin=-28672	PositionMin=-28672	PositionMin=-28672
MinimalWidth=22	MinimalWidth=22	MinimalWidth=22	MinimalWidth=22	MinimalWidth=22
MaximalWidth=4096	MaximalWidth=4096	MaximalWidth=4096	MaximalWidth=4096	MaximalWidth=4096
PositionWidth=200	PositionWidth=200	PositionWidth=200	PositionWidth=200	PositionWidth=200
PositionMax=28672	PositionMax=28672	PositionMax=28672	PositionMax=28672	PositionMax=28672
AngleMin=-7.000	AngleMin=-7.000	AngleMin=-7.000	AngleMin=-7.000	AngleMin=-7.000
AngleWidth=0.0100	AngleWidth=0.0537	AngleWidth=0.1000	AngleWidth=0.0537	*AngleWidth=0.0537*
AngleBias=0.000	AngleBias=0.000	AngleBias=0.000	AngleBias=0.000	AngleBias=0.000
AngleMax=7.000	AngleMax=7.000	AngleMax=7.000	AngleMax=7.000	AngleMax=7.000
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=28672	DeltaPosition=-1641	DeltaPosition=0	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=-2.44140625e-4	Koeff_1=-2.4414e-3	Koeff_1=-2.44140625e-4	Koeff_1=-2.4414e-3	*Koeff_1=-2.4414e-3*
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1
RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=1
MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[Motor5]	[Motor5]	[Motor5]	[Motor5]	[Motor5]
; Entweder X-Achse oder Absorber als Motor 5 aktivieren				
; >>> Umstecken				
Name=Absorber	Name=Absorber	Name=Absorber	Name=Absorber	Name=Absorber
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
MaxVelocity=7000	MaxVelocity=7000	MaxVelocity=7000	MaxVelocity=7000	MaxVelocity=7000
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
Unit=Channel	Unit=Channel	Unit=Channel	Unit=Channel	Unit=Channel
Digits=1	Digits=1	Digits=1	Digits=1	Digits=1
BoardId=3	BoardId=3	BoardId=3	BoardId=3	BoardId=3
DeathBand=2	DeathBand=2	DeathBand=2	DeathBand=2	DeathBand=2
Velocity=6998	Velocity=6999	Velocity=6998	Velocity=6998	Velocity=6998
SpeedScale=180.0	SpeedScale=180.0	SpeedScale=180.0	SpeedScale=180.0	SpeedScale=180.0
Torque=50	Torque=50	Torque=50	Torque=50	Torque=50
Gain=200	Gain=200	Gain=200	Gain=200	Gain=200
DynamicGain=110	DynamicGain=110	DynamicGain=110	DynamicGain=110	DynamicGain=110
Acceleration=6000	Acceleration=6000	Acceleration=6000	Acceleration=6000	Acceleration=6000
DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10
RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000	RemoveLimit=4000
InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1	InitialMove=1
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
DistanceToZero=18941	DistanceToZero=18941	DistanceToZero=18941	DistanceToZero=18941	DistanceToZero=18941
PositionMin=-11000	PositionMin=-11000	PositionMin=-11000	PositionMin=-11000	PositionMin=-11000
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=2000	MaximalWidth=2000	MaximalWidth=2000	MaximalWidth=2000	MaximalWidth=2000
PositionWidth=1799	PositionWidth=1799	PositionWidth=1799	PositionWidth=1799	PositionWidth=1799
PositionMax=225000	PositionMax=225000	PositionMax=225000	PositionMax=225000	PositionMax=225000
AngleMin=-45.1	AngleMin=-45.1	AngleMin=-45.1	AngleMin=-45.1	AngleMin=-45.1
AngleWidth=1.00	AngleWidth=1.00	AngleWidth=1.00	AngleWidth=1.00	AngleWidth=1.00
AngleBias=0.0	AngleBias=0.0	AngleBias=0.0	AngleBias=0.0	AngleBias=0.0
AngleMax=923.5	AngleMax=923.5	AngleMax=923.5	AngleMax=923.5	AngleMax=923.5
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=0	Direction=0	Direction=0	Direction=0	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=0	DeltaPosition=-2022	DeltaPosition=-156000	DeltaPosition=0	*DeltaPosition=0*
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=4.1045454e-3	Koeff_1=4.1045454e-3	Koeff_1=4.1045454e-3	Koeff_1=4.1045454e-3	Koeff_1=4.1045454e-3
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=1	Upwards=1	Upwards=1	Upwards=1	Upwards=1
Position=-1	Position=-1	Position=-1	Position=-1	
AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	
MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0	MaxFailure=30.0
RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=0	RestartPosible=1
Correction=0	Correction=0	Correction=0	Correction=0	Correction=0

devehrm1.ini	devehrm2.ini	Hrm1.ini	hrm2.ini	diff_hardware.ini
[_Motor5]	[_Motor5]	[_Motor5]	[_Motor5]	[_Motor5]
; Entweder X-Achse oder Absorber als Motor 5 aktivieren ; >>> Umstecken				
Name=X	Name=X	Name=X	Name=X	Name=X
Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA	Type=C-812ISA
RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000	RamAddr=0xD000
BoardId=3	BoardId=3	BoardId=3	BoardId=3	BoardId=3
Unit=Millimeter	Unit=Millimeter	Unit=Millimeter	Unit=Millimeter	Unit=Millimeter
MaxVelocity=84	MaxVelocity=84	MaxVelocity=84	MaxVelocity=84	MaxVelocity=84
DeathBand=9	DeathBand=1	DeathBand=9	DeathBand=1	*DeathBand=1*
Digits=2	Digits=2	Digits=2	Digits=2	Digits=2
Velocity=84	Velocity=84	Velocity=84	Velocity=84	Velocity=84
SpeedScale=10.0	SpeedScale=10.0	SpeedScale=10.0	SpeedScale=10.0	SpeedScale=10.0
Torque=120	Torque=120	Torque=120	Torque=120	Torque=120
Gain=200	Gain=200	Gain=200	Gain=200	Gain=200
DynamicGain=150	DynamicGain=150	DynamicGain=150	DynamicGain=150	DynamicGain=150
Acceleration=1600	Acceleration=1600	Acceleration=1600	Acceleration=1600	Acceleration=1600
DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10	DeccelerationPoint=10
InitialMove=0	InitialMove=0	InitialMove=0	InitialMove=0	InitialMove=0
RemoveLimit=600	RemoveLimit=600	RemoveLimit=600	RemoveLimit=600	RemoveLimit=600
IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0	IndexLine=0
MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0	MoveFirstToLimit=0
InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0	InitialAngle=0.0
DistanceToZero=0	DistanceToZero=0	DistanceToZero=0	DistanceToZero=0	DistanceToZero=0
PositionMin=-4000	PositionMin=-4000	PositionMin=-4000	PositionMin=-4000	PositionMin=-4000
MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2	MinimalWidth=2
MaximalWidth=800	MaximalWidth=800	MaximalWidth=800	MaximalWidth=800	MaximalWidth=800
PositionWidth=200	PositionWidth=200	PositionWidth=200	PositionWidth=200	PositionWidth=200
PositionMax=4000	PositionMax=4000	PositionMax=4000	PositionMax=4000	PositionMax=4000
AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00	AngleMin=-10.00
AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100	AngleWidth=0.100
AngleBias=0.00	AngleBias=0.00	AngleBias=0.00	AngleBias=0.00	AngleBias=0.00
AngleMax=10.00	AngleMax=10.00	AngleMax=10.00	AngleMax=10.00	AngleMax=10.00
Orientation=1	Orientation=1	Orientation=1	Orientation=1	
Direction=1	Direction=1	Direction=1	Direction=1	
Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0	Hysteresis=0
DeltaPosition=-1	DeltaPosition=-1	DeltaPosition=-1	DeltaPosition=-1	DeltaPosition=0
Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0	Koeff_3=0.0
Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0	Koeff_2=0.0
Koeff_1=2.500e-03	Koeff_1=2.500e-03	Koeff_1=2.500e-03	Koeff_1=2.500e-03	Koeff_1=2.500e-03
Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	Koeff_0=0.0	
Upwards=0	Upwards=0	Upwards=0	Upwards=0	Upwards=0
Position=-1	Position=-1	Position=-1	Position=-1	
AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	AngleZero=0.0	
RestartPosible=1	RestartPosible=1	RestartPosible=1	RestartPosible=1	RestartPosible=1

C.2.3 Bedeutung der Hardware-Parameter

INI Parameter	Repräsentant im Programm	Werte	Bedeutung für Typen	Verwendung im XCTL-System				Bedeutung in Umgebungssimulation
				INI Lesen	INI Modif.	Repr. Lesen	Repr. Modif.	
[MotorX]								
Type	ctype (temporär)	bekannte Typen: [C-812ISA | C-812GPIB | C-832 | TMotor] wenn unbekannt -> Tmotor	-	x	-	x	-	x
Name	szCharacteristic	i.a. beliebig; einige vordefiniert zur Rollenverteilung (Omega, Theta, Psi ...)	C-832, C-812	x	-	x	-	x
BoardId	nOnBoardId	[0 | 1] (C-832) [1 .. 4] (C-812ISA, C-812GBIB)	C-832, C-812	x	-	x	-	x
RamAddr	wBaseAddr	0xC800 ... 0xDE00	C-812	x	-	x	-	x
IOAddr	wBaseAddr	0x0200 ...	C-832	x	-	x	-	x
Hysteresis	dwHysteresis	[0 .. ?]	C-832, C-812	x	-	x	-	x
Upwards	bUpwards	[0 | 1]	C-832, C-812	x	x	x	x	x
MaxVelocity	dwMaxVelocity	[0 .. 8.388.680] (C812) [0 .. 1.073.741.823 = 0x3FFFFFFF] (C832)	C-832, C-812	x	x	x	x	x
Velocity	dwVelocity	[1 .. MaxVelocity]	C-832, C-812	x	x	x	x	x
Unit	eUnit	i.a. beliebig; einige vordefiniert zur Umrechnung im Programm: [Grad | Minuten | Sekunden | Millimeter | Mikrometer]	C-832, C-812	x	-	x	-	x
DigIts	DFmt, SFmt	[0 .. ?]	C-832, C-812	x	-	x	-	x
MinimalWidth	wPositionMinWidth	[0 .. ?]	C-832, C-812	x	-	x	-	x
MaximalWidth	wPositionMaxWidth	[0 .. ?]	C-832, C-812	x	-	x	-	x
PositionMin	lPositionMin	[? .. ?]	C-832, C-812	x	-	x	-	x
PositionMax	lPositionMax	[? .. ?]	C-832, C-812	x	-	x	x wenn Max<Min	x

PositionWidth	wPositionWidth	real, >0	C-832, C-812	x	x	x	x	x	x
InitialMove	bInitialMoveReady	[0 | 1]	C-832, C-812	x	-	x	-		x
RestartPosible	bIsCalibrated	[0 | 1]	C-832, C-812	x	x	x	x		x
DeltaPosition	lDeltaPosition	[? .. ?]	C-832, C-812	x	x aus IPosition	x	x		x
DistanceToZero	lDistanceToZero	[? .. ?]	C-832, C-812	x	x	x	x		x
RemoveLimit	dwRemoveLimit	[0 .. ?]	C-832, C-812	x	-	x	-		x
MoveFirstToLimit	bMoveFirstToLimit	[0 | 1]	C-832	x	-	x	-	nur Einfluss, wenn IndexLine=1	x
IndexLine	bIndexLine	[0 | 1]	C-832	x	-	x	x		x
InitialAngle	dInitialAngle	real	C-832, C-812	x	-	x	-		x
Acceleration	dwAcceleration	[0 .. 8.388.680] (C812) [0 .. 1.073.741.823 = 0x3FFFFFFF] (C832)	C-832, C-812	x	x	x	x	TOptimizeDC_832::Dlg_OnCommand();	x
AngleMin	dAngleMin	[? .. ?]	C-832, C-812	x	x	x	x		x
AngleMax	dAngleMax	[? .. ?]	C-832, C-812	x	x	x	x		x
AngleWidth	dAngleWidth	[? .. ?]	C-832, C-812	x	x	x	x		x
AngleBias	dAngleBias	[? .. ?]	C-832, C-812	-	x	x	x	obsolet, da nicht aus INI-File gelesen	
Koeff_1 Koeff_2 Koeff_3	dKoeff_1 dKoeff_2 dKoeff_3	real	C-832, C-812	x	-	x	-	Koeff_2, Koeff_3 nur wenn Correction=1	x
SpeedScale	fSpeedScale	real, >0	C-832, C-812	x	-	x	-		x
MaxFailure	MaxFailure	real	C-832, C-812	x	-	x	-	nur wenn Correction=1 und Koeff_2, Koeff_3 definiert	x
Correction	bCorrection	[0 | 1]	C-832, C-812	x	-	x	-		x
DeathBand	wDeathBand	[1 .. 126]	C-812	x	-	x	-	TC_812ISA::isMoveFinish();	x
DeccelerationPoint	wDeccelerationPoint	[0 .. ?]	C-812	x	-	x	-		-

Torque	wTorque	[1 .. 127]	C-812	x	x	x	x	-
Gain	wStaticGain (C812), wKP (C832)	[1 .. 255] (C812) [1 .. 32767] (C832)	C-832, C-812	x	x	x	x	-
DynamicGain	wDynamicGain (C812), wKD (C832)	[1 .. 255] (C812) [1 .. 32767] (C832)	C-832, C-812	x	x	x	x	-
IntegralGain	wKI	[1 .. 32767]	C-832	x	x	x	x	-
IntegralLimit	wKL	[1 .. 32767]	C-832	x	x	x	x	-
DifferentialEncoder	-	[0 | 1]	C-832	x	-	-	-	? Konfigurationsregister (cConfig) wird modifiziert
EnableInterrupts	-	[0 | 1]	C-832	x	-	-	-	? Konfigurationsregister (cConfig) wird modifiziert
GPIBAddr	wGPIBAddr	[0 .. ?]	C-812GPIB	x	-	x	-	GPIB nicht unterstützt !
InquireStatus	bInquireStatus	[0 | 1]	C-812GPIB	x	-	x	-	GPIB nicht unterstützt !
MoveFirstToZero	-	-	-	-	-	-	-	obsolet
Orientation	-	-	-	-	-	-	-	obsolet
Direction	-	-	-	-	-	-	-	obsolet
Position	-	-	-	-	-	-	-	obsolet
AngleZero	-	-	-	-	-	-	-	obsolet
Koeff_0	-	-	-	-	-	-	-	obsolet

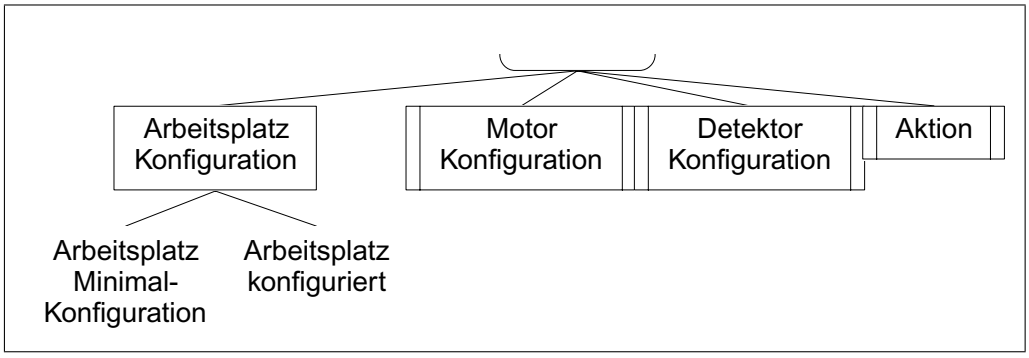
INI Parameter	Repräsentant im Programm	Werte	Bedeutung für Typen	Verwendung im XCTL-System				Bedeutung in Umgebungssimulation
				INI Lesen	INI Modif.	Repr. Lesen	Repr. Modif.	
[DeviceX]								
Type	buf (temporär)	bekannte Typen: [Generic | Stoe-Psd | Radicon | Matrox | Braun-Psd | Psd | Encoder | Test] wenn unbekannt -> TDevice-Klasse	-	x	-	x	-	Generic, Radicon, Braun-Psd, Encoder nicht unterstützt ! Stoe-Psd keine simulierte Intensität. Matrox nicht implementiert (CCD).
Name	Characteristic	beliebig, bis auf Ausnahme: Detektor vom Typ Test sollte "Counter" heissen -> notwendig für Autom. Justage	alle	x	-	x	-	x
Sound	bSound	[0 | 1]	für alle vorgesehen (benutzt von Radicon, Generic)	x	x	x	x	-
Debug	bDebug	[0 | 1]	für alle vorgesehen (benutzt von Braun-Psd, Psd, Stoe-Psd, Radicon, unbekannt)	x	-	x	-	x
ExposureTime	fExposureTime	[0.1 .. 500.0]	alle	x	x	x	x	x
ExposureCounts	dwExposureCounts	[10 .. ?]	alle	x	x	x	x	-
Failure	fFailure	real	alle	-	-	x	x	nicht unterstützt, aber in Device-Dialog einstellbar ! (Mess-Fehler wird nicht simuliert)
Unit	eUnit	i.a. beliebig; einige vordefiniert zur Umrechnung im Programm: [Grad | Minuten | Sekunden | Millimeter | Mikrometer]	Psd's	x	-	x	-	x (Einheit für AngleStep)
AngleStep	fAngleStep	real, >0	Psd's	x	x	x	x	x
HVRegelung	bHVRegelung	[0 | 1]	Psd's	x	x	x	x	-
ReadLeftFirst	bReadLeftFirst	[0 | 1]	Psd's	x	x	x	x	x

SignalGrowUp	bSignalGrowUp	[0 | 1]	Psd, Stoe-Psd	x	-	x	x	Wann wird das verwendet ? Jede Funktion setzt diesen Parameter selbst.
AddedChannels	nAddedChannels	[1 .. 4096]	Psd's	x	x	x	x	x
FirstChannel	nMinChannel	[0 .. ?]	für alle Psd's vorgesehen (benutzt von Stoe-Psd, Braun-Psd)	x	x	x	x	-
LastChannel	nMaxChannel	[0 .. ?]	für alle Psd's vorgesehen (benutzt von Stoe-Psd, Braun-Psd)	x	x	x	x	-
OverflowIntensity	OverflowIntensity	real, >0	Psd's	x	-	x	-	x
BaseAddr	nBaseAddr	[0x100 .. 0x3F8] (4b-Schritte)	Braun-Psd, Stoe-Psd	x	-	x	-	-

Anhang D

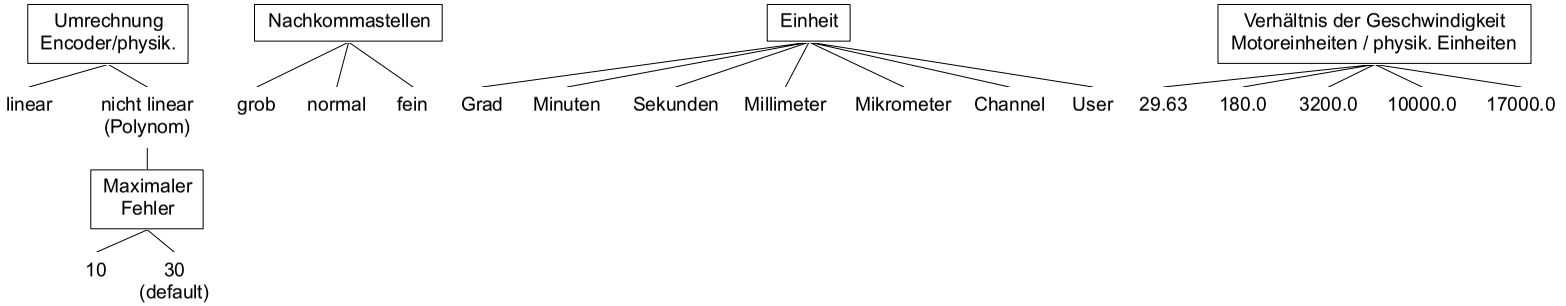
Klassifikationsbäume „LineScan“

Unter Rücksichtnahme auf die Seitenzahl dieser Arbeit, wird auf die Abbildung der Kombinationstabellen zur Spezifikation von Testsequenzen verzichtet.

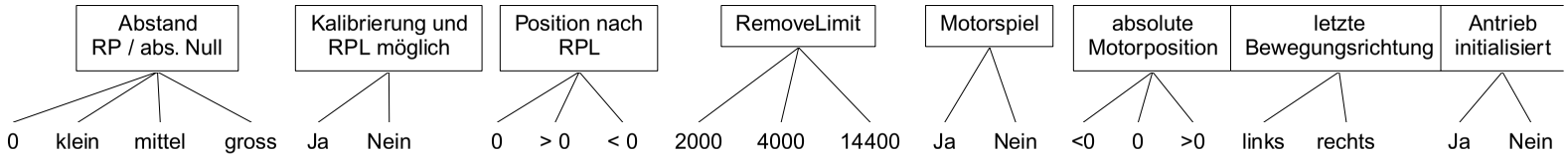


LineScan /Motor Konfiguration
Allgemeine und Typenabhängige Parameter
Parameter zur Einheitenumrechnung
Parameter zum Referenzpunkt und zur Positionsbestimmung
Bewegungs-Parameter und Software-Schranken
Ansteuerungs-Parameter (keine Auswirkung in Umgebungssimulation)
LineScan /Motor Konfiguration /Allgemeine und Typenabhängige Parameter
Motortyp
C-812ISA
C-832
BoardID
1 2 3 4
Ram Adresse
0xC800 0xD000 0xDE00
Zulässiger Positionsfehler
klein mittel gross max
BoardID
0 1
Port Adresse
0x200 0x210 0x300
differentielle Encodersignale unterstützen
Ja Nein
Interrupts
Enable Disable
Index-Schalter vorhanden
Ja Nein
beim Anfahren der Indexposition
vorher linke Hardwareschranke anfahren
gleich zur Indexposition
Achse
Theta Omega Psi Phi Y Absorber Kollimator

LineScan /Motor Konfiguration /Parameter zur Einheitenumrechnung

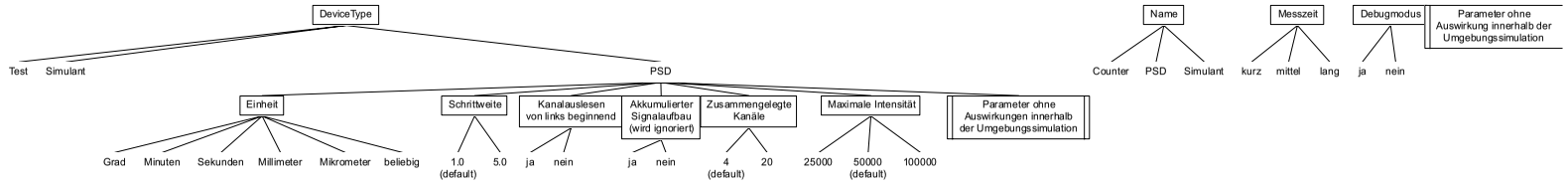


LineScan /Motor Konfiguration /Parameter zum Referenzpunkt und zur Positionsbestimmung

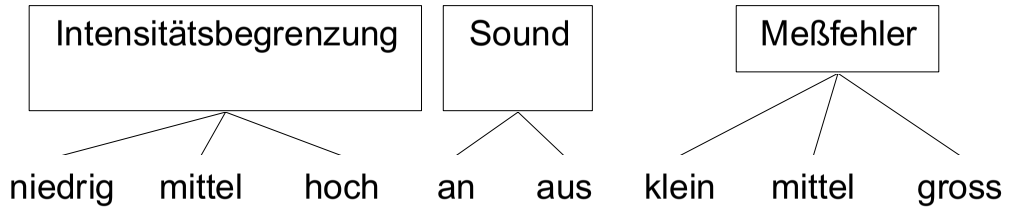


LineScan /Motor Konfiguration /Bewegungs-Parameter und Software-Schranken
absolute Position der Softwareschranken
links
-1400000 -30000 -20000
Schrittweite
20 150 200 750 2000
rechts
20000 30000 200000
Schranken der Schrittweite
links
2 4 22
rechts
500 4000 20000
Nutzerbereich
Schranke links
-1000.0 -45.0 -10.0 -7.0
Schrittweite
0.005 0.05 0.1 0.5 1.0
Schranke rechts
7.0 10.0 100.0 1000.0
Geschwindigkeit
klein normal groß
Beschleunigung
klein normal groß
LineScan /Motor Konfiguration /Ansteuerungs-Parameter (keine Auswirkung in Umgebungssimulation)
maximaler Motorstrom (C812)
50 80 120
Verstärkung
statische
180 200 400 600
dynamische
10 60 90 110 150
DeccelerationPoint (C812)
10 20 (default)
integral (C832)
Gain
5 10
Limit
300 4000

LineScan /Detektor Konfiguration



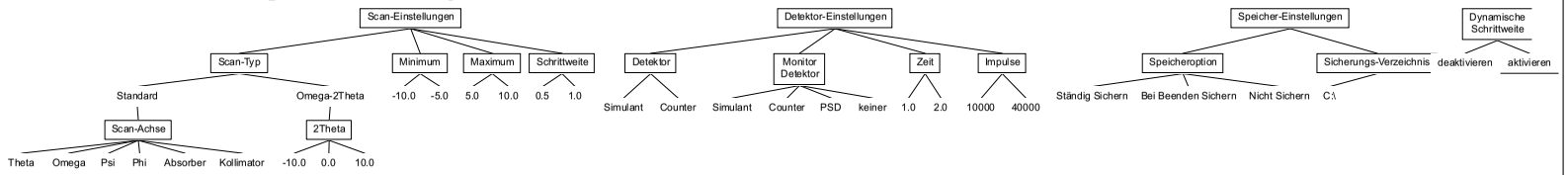
LineScan /Detektor Konfiguration /Parameter ohne Auswirkung innerhalb der Umgebungssimulation



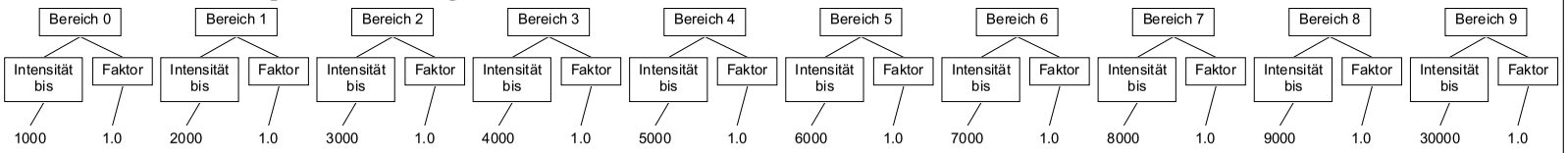
LineScan /Detektor Konfiguration /Parameter ohne Auswirkungen innerhalb der Umgebungssimulation
Hochspannungs-steuerung
erster benutzter Kanal
letzter benutzter Kanal
ja
nein
0 (default)
2000
4095
0
2000
4095 (default)

LineScan /Aktion
Umgebung
Programm
Aktion
Starten
Beenden
Dateiverwaltung
Was
Neu
Nachladen
Sichern unter
Sichern
Fenster LineScan
Aktion
Öffnen
Schliessen
StepScan starten
Scan unterbrechen bzw. weiterführen
Messkurve bearbeiten
Wie
Fixieren
Speichern
Loeschen
Dialogbox StepScan
Aktion
Öffnen
Schliessen
Messparameter festlegen
Wie
Ok
Abbrechen
Filedialogbox
Typ
Öffnen
Speichern
Messageboxen behandeln
Typ
Meldung
Frage
Start Scan
Protokollbuch Meldung
Schliessen mit
OK
X
Schliessen mit
JA
NEIN
X
Schliessen mit
JA
NEIN
X
Schliessen mit
JA
NEIN
X
Zeit Abwarten

LineScan /Aktion /Messparameter festlegen



LineScan /Aktion /Messparameter festlegen /aktivieren



Anhang E

Quellcode-Instrumentierung

E.1 MAKEFILE

```
#------------------------------------------------------------------------------
#
# Borland C++ IDE generated makefile for project XCTL
# modified for tool CTC++ (Testwell Oy) for code instrumentation
# (instrumentation with full qualification of all functions)
#
# Jens Hanisch, 08.08.02
#
# XCTL CVS-Version 20.01.2002
#

.AUTODEPEND

xcontrol :

#------------------------------------------------------------------------------
#
# Borland C++ tools
#
IMPLIB  = Implib
BCC     = Bcc +BccW16.cfg
TLINK   = TLink
TLIB    = TLib
BRC     = Brc
TASM    = Tasm



#------------------------------------------------------------------------------
#
# External tools
#
CompileHLP = hc31.exe  #IDE Command Line: $NOSWAP $CAP MSG(HC312MSG) $EDNAME



#------------------------------------------------------------------------------
#
# Macros having a default
```

```
#
!ifndef MONITOR
MONITOR     = MON
!endif

!ifndef IMODE
IMODE       = f
!endif

!ifndef CTCHOME
CTCHOME     = G:\Studium\Diplom\Tools\CTC
!endif

!ifndef XCTLHOME
XCTLHOME    = G:\Studium\Diplom\XCTL\EXE
!endif

!ifndef INCLUDE
INCLUDE     = F:\PROGRA~1\BC45\INCLUDE;.\INCLUDE;$(CTCHOME)
!endif

!ifndef LIBS
LIBS        = F:\PROGRA~1\BC45\LIB;.\BC4LIBS
!endif

!ifndef CTCINIT
CTCINIT     = ctc.ini
CTCCOPT     = -c $(CTCINIT)
!endif

!ifndef DATAFILE
DATAFILE    = $(XCTLHOME)\$(MONITOR).dat
!endif

CTCDIR      = ctc.dir

CTCFLAGS    = -name $(MONITOR) -o $(CTCDIR) -i $(IMODE) -update -skipinclude \
              -c $(CTCINIT) -datafile $(DATAFILE)



##############################################################################
#
# Probleme und Konflikte bei der Instrumentierung der XCTL-Quellen
# ================================================================
#
#Datei                       Problem / Konflikt         Loesung
#----------------------------------------------------------------------------
#
#DETECUSE\KISL1.C            K&R 1 Funktionen werden    Typenvereinbarung
#                            von CTC++ nicht            in Liste der formalen
#                            unterstützt !              Parameter schreiben
#                                                       (K&R 2/ANSI-Norm)
#
#DETECUSE\KMPT1.C            Assemblercode in Fkt.      Einfuegen von
#                            in_byte() fehlerhaft       Semikolons nach jedem
#                            nach Instrumentierung      Assemblerbefehl
#



##############################################################################
```

```
#
# Target for instrumentation
# (with Precompiler-Macros)
#
ctcinstrument: $(CTCDIR)
    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- AUTOJUST\M_JUSTAG.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- AUTOJUST\MATRIX.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- AUTOJUST\TRANSFRM.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib)
| -- DATAVISA\M_CURVE.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- DATAVISA\M_DATA.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_detecusebam9513dcpp) $(CNIEAT_detecusebam9513dcpp)
| -- DETECUSE\AM9513.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_detecusebbraunpsddcpp) $(CNIEAT_detecusebbraunpsddcpp)
| -- DETECUSE\BRAUNPSD.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib)
| -- DETECUSE\C_LAYER.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_detecusebcountersdcpp) $(CNIEAT_detecusebcountersdcpp)
| -- DETECUSE\COUNTERS.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags @@&&|
```

```
$(CEAT_detecusebkisl1dc) $(CNIEAT_detecusebkisl1dc)
| -- -cxxflags -- DETECUSE\KISL1.C

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags @@&&|
$(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib)
| -- -cxxflags -- DETECUSE\KMPT1.C

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib)
| -- DETECUSE\TESTDEV.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- DIFRKMTY\M_ARSCAN.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- DIFRKMTY\M_CCDSCN.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- DIFRKMTY\M_SCAN.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- DIFRKMTY\M_dlgdif.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib)
| -- INTERNLS\L_LAYER.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- INTERNLS\M_MAIN.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_motrstrgbm_layerdcpp) $(CNIEAT_motrstrgbm_layerdcpp)
| -- MOTRSTRG\M_LAYER.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebMotorsdlib) $(CNIEAT_exebMotorsdlib)
| -- MOTRSTRG\MOTORS.CPP
```

```
    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- MOTRSTRG\MSIMSTAT.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_swintracbdlg_tpldcpp_CO) $(CNIEAT_swintracbdlg_tpldcpp)
| -- SWINTRAC\DLG_TPL.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- SWINTRAC\M_DEVICE.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_swintracbm_dlgdcpp) $(CNIEAT_swintracbm_dlgdcpp)
| -- SWINTRAC\M_DLG.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- TOPOGRFY\M_TOPO.CPP

    $(CTCHOME)\ctc @&&|
$(CTCFLAGS)
| -cflags -- -cxxflags @@&&|
$(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe)
| -- WORKFLOW\M_STEERG.CPP



################################################################################
#
# Targets for copy
# (without instrumentation)
#
ctccopy: $(CTCDIR)
#   Copy AUTOJUST\M_JUSTAG.CPP $(CTCDIR)
#   Copy AUTOJUST\MATRIX.CPP $(CTCDIR)
#   Copy AUTOJUST\TRANSFRM.CPP $(CTCDIR)
#   Copy DATAVISA\M_CURVE.CPP $(CTCDIR)
#   Copy DATAVISA\M_DATA.CPP $(CTCDIR)
#   Copy DETECUSE\AM9513.CPP $(CTCDIR)
#   Copy DETECUSE\BRAUNPSD.CPP $(CTCDIR)
#   Copy DETECUSE\C_LAYER.CPP $(CTCDIR)
#   Copy DETECUSE\COUNTERS.CPP $(CTCDIR)
#   Copy DETECUSE\KISL1.C $(CTCDIR)
#   Copy DETECUSE\KMPT1.C $(CTCDIR)
#   Copy DETECUSE\TESTDEV.CPP $(CTCDIR)
#   Copy DIFRKMTY\M_ARSCAN.CPP $(CTCDIR)
#   Copy DIFRKMTY\M_CCDSCN.CPP $(CTCDIR)
#   Copy DIFRKMTY\M_SCAN.CPP $(CTCDIR)
#   Copy DIFRKMTY\M_dlgdif.CPP $(CTCDIR)
#   Copy INTERNLS\L_LAYER.CPP $(CTCDIR)
```

```
#   Copy INTERNLS\M_MAIN.CPP $(CTCDIR)
#   Copy MOTRSTRG\M_LAYER.CPP $(CTCDIR)
#   Copy MOTRSTRG\MOTORS.CPP $(CTCDIR)
#   Copy MOTRSTRG\MSIMSTAT.CPP $(CTCDIR)
#   Copy SWINTRAC\DLG_TPL.CPP $(CTCDIR)
#   Copy SWINTRAC\M_DEVICE.CPP $(CTCDIR)
#   Copy SWINTRAC\M_DLG.CPP $(CTCDIR)
#   Copy TOPOGRFY\M_TOPO.CPP $(CTCDIR)
#   Copy WORKFLOW\M_STEERG.CPP $(CTCDIR)


################################################################################
#
# Create directories for object files
#
!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF

$(CTCDIR):
        if not exist OBJ/$(NULL) mkdir OBJ
        if not exist $(CTCDIR)/$(NULL) mkdir $(CTCDIR)
        if not exist $(CTCDIR)\OBJ/$(NULL) mkdir $(CTCDIR)\OBJ

objectdir:
        if not exist OBJ/$(NULL) mkdir OBJ



################################################################################
#
# Analysing monitor
#
lst:
        $(CTCHOME)\ctcpost @&&|
$(CTCCOPT) -n $(CTCDIR)\$(MONITOR) -d $(DATAFILE) \
-p profile.lst -s summary.lst -u untested.lst
|

htmprofile:
        -$(CTCHOME)\perl\perl $(CTCHOME)\ctc2html.pl -i profile.lst
        $(CTCHOME)\perl\perl $(CTCHOME)\ctc2html.pl -i profile.lst

htmsummary:
        $(CTCHOME)\perl\perl $(CTCHOME)\ctc2html.pl -i summary.lst

htmuntested:
        $(CTCHOME)\perl\perl $(CTCHOME)\ctc2html.pl -i untested.lst



################################################################################
#
# Options (Borland C++ IDE generated makefile for project XCTL)
#
IDE_LFLAGS =  -i -f -Gr -L$(LIBS) -e -3 -A=16
IDE_RFLAGS =  -31 -k

CW16Entwickeln_EXE =  -WS
```

```
LW16Entwickeln_EXE =  -c -C
RW16Entwickeln_EXE =
BW16Entwickeln_EXE =

CW16Entwickeln_DLL =  -WD -x-
LW16Entwickeln_DLL =  -c -C
RW16Entwickeln_DLL =
BW16Entwickeln_DLL =

CNATW16_exebSPLibdlib = $(CW16Entwickeln_DLL)
LNATW16_exebSPLibdlib = $(LW16Entwickeln_DLL)
RNATW16_exebSPLibdlib = $(RW16Entwickeln_DLL)
BNATW16_exebSPLibdlib = $(BW16Entwickeln_DLL)
CLATW16_exebSPLibdlib =  -ml -WD -H=SPLIB.PRE -Ff=20000 -h
LLATW16_exebSPLibdlib =  -Twd -c -C
RLATW16_exebSPLibdlib =  -31
BLATW16_exebSPLibdlib =
CEAT_exebSPLibdlib = $(CNATW16_exebSPLibdlib) $(CLATW16_exebSPLibdlib)
CNIEAT_exebSPLibdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Library
LNIEAT_exebSPLibdlib = -x
LEAT_exebSPLibdlib = $(LNATW16_exebSPLibdlib) $(LLATW16_exebSPLibdlib)
REAT_exebSPLibdlib = $(RNATW16_exebSPLibdlib) $(RLATW16_exebSPLibdlib)
BEAT_exebSPLibdlib = $(BNATW16_exebSPLibdlib) $(BLATW16_exebSPLibdlib)

CNATW16_exebMotorsdlib = $(CW16Entwickeln_DLL)
LNATW16_exebMotorsdlib = $(LW16Entwickeln_DLL)
RNATW16_exebMotorsdlib = $(RW16Entwickeln_DLL)
BNATW16_exebMotorsdlib = $(BW16Entwickeln_DLL)
CLATW16_exebMotorsdlib =  -H=MOTOR.PRE -ml -WD
LLATW16_exebMotorsdlib =  -v -Twd -c -C
RLATW16_exebMotorsdlib =  -31
BLATW16_exebMotorsdlib =
CEAT_exebMotorsdlib = $(CNATW16_exebMotorsdlib) $(CLATW16_exebMotorsdlib)
CNIEAT_exebMotorsdlib = -I$(INCLUDE) -D_RTLDLL;_RTLDLL;_BIDSDLL;EX_Build_Motors;Use_Library
LNIEAT_exebMotorsdlib = -x
LEAT_exebMotorsdlib = $(LNATW16_exebMotorsdlib) $(LLATW16_exebMotorsdlib)
REAT_exebMotorsdlib = $(RNATW16_exebMotorsdlib) $(RLATW16_exebMotorsdlib)
BEAT_exebMotorsdlib = $(BNATW16_exebMotorsdlib) $(BLATW16_exebMotorsdlib)

CLATW16_motrstrgbm_layerdcpp =
LLATW16_motrstrgbm_layerdcpp =
RLATW16_motrstrgbm_layerdcpp =
BLATW16_motrstrgbm_layerdcpp =
CEAT_motrstrgbm_layerdcpp = $(CEAT_exebMotorsdlib) $(CLATW16_motrstrgbm_layerdcpp)
CNIEAT_motrstrgbm_layerdcpp = -I$(INCLUDE) -D_RTLDLL;_RTLDLL;_BIDSDLL;EX_Build_Motors;Use_Library
LNIEAT_motrstrgbm_layerdcpp = -x
LEAT_motrstrgbm_layerdcpp = $(LEAT_exebMotorsdlib) $(LLATW16_motrstrgbm_layerdcpp)
REAT_motrstrgbm_layerdcpp = $(REAT_exebMotorsdlib) $(RLATW16_motrstrgbm_layerdcpp)
BEAT_motrstrgbm_layerdcpp = $(BEAT_exebMotorsdlib) $(BLATW16_motrstrgbm_layerdcpp)

CNATW16_exebCountersdlib = $(CW16Entwickeln_DLL)
LNATW16_exebCountersdlib = $(LW16Entwickeln_DLL)
RNATW16_exebCountersdlib = $(RW16Entwickeln_DLL)
BNATW16_exebCountersdlib = $(BW16Entwickeln_DLL)
CLATW16_exebCountersdlib =  -H=COUNTER.PRE -WD -x
LLATW16_exebCountersdlib =  -v -Twd -c -C
RLATW16_exebCountersdlib =  -31
BLATW16_exebCountersdlib =
CEAT_exebCountersdlib = $(CNATW16_exebCountersdlib) $(CLATW16_exebCountersdlib)
CNIEAT_exebCountersdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_exebCountersdlib = -x
LEAT_exebCountersdlib = $(LNATW16_exebCountersdlib) $(LLATW16_exebCountersdlib)
```

```
REAT_exebCountersdlib = $(RNATW16_exebCountersdlib) $(RLATW16_exebCountersdlib)
BEAT_exebCountersdlib = $(BNATW16_exebCountersdlib) $(BLATW16_exebCountersdlib)

CLATW16_swintracbdlg_tpldcpp =  -w-sig
LLATW16_swintracbdlg_tpldcpp =
RLATW16_swintracbdlg_tpldcpp =
BLATW16_swintracbdlg_tpldcpp =

CEAT_swintracbdlg_tpldcpp = $(CEAT_exebSPLibdlib) $(CLATW16_swintracbdlg_tpldcpp)
CNIEAT_swintracbdlg_tpldcpp = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Library
LNIEAT_swintracbdlg_tpldcpp = -x
LEAT_swintracbdlg_tpldcpp = $(LEAT_exebSPLibdlib) $(LLATW16_swintracbdlg_tpldcpp)
REAT_swintracbdlg_tpldcpp = $(REAT_exebSPLibdlib) $(RLATW16_swintracbdlg_tpldcpp)
BEAT_swintracbdlg_tpldcpp = $(BEAT_exebSPLibdlib) $(BLATW16_swintracbdlg_tpldcpp)

CLATW16_detecusebcountersdcpp =
LLATW16_detecusebcountersdcpp =
RLATW16_detecusebcountersdcpp =
BLATW16_detecusebcountersdcpp =
CEAT_detecusebcountersdcpp = $(CEAT_exebCountersdlib) $(CLATW16_detecusebcountersdcpp)
CNIEAT_detecusebcountersdcpp = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_detecusebcountersdcpp = -x
LEAT_detecusebcountersdcpp = $(LEAT_exebCountersdlib) $(LLATW16_detecusebcountersdcpp)
REAT_detecusebcountersdcpp = $(REAT_exebCountersdlib) $(RLATW16_detecusebcountersdcpp)
BEAT_detecusebcountersdcpp = $(BEAT_exebCountersdlib) $(BLATW16_detecusebcountersdcpp)

CLATW16_detecusebam9513dcpp =
LLATW16_detecusebam9513dcpp =
RLATW16_detecusebam9513dcpp =
BLATW16_detecusebam9513dcpp =
CEAT_detecusebam9513dcpp = $(CEAT_exebCountersdlib) $(CLATW16_detecusebam9513dcpp)
CNIEAT_detecusebam9513dcpp = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_detecusebam9513dcpp = -x
LEAT_detecusebam9513dcpp = $(LEAT_exebCountersdlib) $(LLATW16_detecusebam9513dcpp)
REAT_detecusebam9513dcpp = $(REAT_exebCountersdlib) $(RLATW16_detecusebam9513dcpp)
BEAT_detecusebam9513dcpp = $(BEAT_exebCountersdlib) $(BLATW16_detecusebam9513dcpp)

CLATW16_detecusebbraunpsddcpp =
LLATW16_detecusebbraunpsddcpp =
RLATW16_detecusebbraunpsddcpp =
BLATW16_detecusebbraunpsddcpp =
CEAT_detecusebbraunpsddcpp = $(CEAT_exebCountersdlib) $(CLATW16_detecusebbraunpsddcpp)
CNIEAT_detecusebbraunpsddcpp = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_detecusebbraunpsddcpp = -x
LEAT_detecusebbraunpsddcpp = $(LEAT_exebCountersdlib) $(LLATW16_detecusebbraunpsddcpp)
REAT_detecusebbraunpsddcpp = $(REAT_exebCountersdlib) $(RLATW16_detecusebbraunpsddcpp)
BEAT_detecusebbraunpsddcpp = $(BEAT_exebCountersdlib) $(BLATW16_detecusebbraunpsddcpp)

CLATW16_detecusebkisl1dc =  -w-sig
LLATW16_detecusebkisl1dc =
RLATW16_detecusebkisl1dc =
BLATW16_detecusebkisl1dc =
CEAT_detecusebkisl1dc = $(CEAT_exebCountersdlib) $(CLATW16_detecusebkisl1dc)
CNIEAT_detecusebkisl1dc = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_detecusebkisl1dc = -x
LEAT_detecusebkisl1dc = $(LEAT_exebCountersdlib) $(LLATW16_detecusebkisl1dc)
REAT_detecusebkisl1dc = $(REAT_exebCountersdlib) $(RLATW16_detecusebkisl1dc)
BEAT_detecusebkisl1dc = $(BEAT_exebCountersdlib) $(BLATW16_detecusebkisl1dc)

CNATW16_exebDevelopdexe = $(CW16Entwickeln_EXE)
LNATW16_exebDevelopdexe = $(LW16Entwickeln_EXE)
RNATW16_exebDevelopdexe = $(RW16Entwickeln_EXE)
```

```
BNATW16_exebDevelopdexe = $(BW16Entwickeln_EXE)
CLATW16_exebDevelopdexe =  -Od -H=RTK_DEV.CSM -ml -WS
LLATW16_exebDevelopdexe =  -v -Twe -c -C
RLATW16_exebDevelopdexe =  -31
BLATW16_exebDevelopdexe =
CEAT_exebDevelopdexe = $(CNATW16_exebDevelopdexe) $(CLATW16_exebDevelopdexe)
CNIEAT_exebDevelopdexe = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;WIN31;Use_Counters;Use_ImageCCD;Use_Library
LNIEAT_exebDevelopdexe = -x
LEAT_exebDevelopdexe = $(LNATW16_exebDevelopdexe) $(LLATW16_exebDevelopdexe)
REAT_exebDevelopdexe = $(RNATW16_exebDevelopdexe) $(RLATW16_exebDevelopdexe)
BEAT_exebDevelopdexe = $(BNATW16_exebDevelopdexe) $(BLATW16_exebDevelopdexe)

CLATW16_swintracbm_dlgdcpp =
LLATW16_swintracbm_dlgdcpp =
RLATW16_swintracbm_dlgdcpp =
BLATW16_swintracbm_dlgdcpp =
CEAT_swintracbm_dlgdcpp = $(CEAT_exebDevelopdexe) $(CLATW16_swintracbm_dlgdcpp)
CNIEAT_swintracbm_dlgdcpp = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;WIN31;Use_Counters;Use_ImageCCD;Use_Library
LNIEAT_swintracbm_dlgdcpp = -x
LEAT_swintracbm_dlgdcpp = $(LEAT_exebDevelopdexe) $(LLATW16_swintracbm_dlgdcpp)
REAT_swintracbm_dlgdcpp = $(REAT_exebDevelopdexe) $(RLATW16_swintracbm_dlgdcpp)
BEAT_swintracbm_dlgdcpp = $(BEAT_exebDevelopdexe) $(BLATW16_swintracbm_dlgdcpp)

############################################################################
#
#mehrfach (nicht benutzt!)
#
CLATW16_Objbsplibdlib =
LLATW16_Objbsplibdlib =
RLATW16_Objbsplibdlib =
BLATW16_Objbsplibdlib =

CEAT_Objbsplibdlib = $(CEAT_exebMotorsdlib) $(CLATW16_Objbsplibdlib)
CNIEAT_Objbsplibdlib = -I$(INCLUDE) -D_RTLDLL;_RTLDLL;_BIDSDLL;EX_Build_Motors;Use_Library
LNIEAT_Objbsplibdlib = -x
LEAT_Objbsplibdlib = $(LEAT_exebMotorsdlib) $(LLATW16_Objbsplibdlib)
REAT_Objbsplibdlib = $(REAT_exebMotorsdlib) $(RLATW16_Objbsplibdlib)
BEAT_Objbsplibdlib = $(BEAT_exebMotorsdlib) $(BLATW16_Objbsplibdlib)

CEAT_Objbsplibdlib = $(CEAT_exebCountersdlib) $(CLATW16_Objbsplibdlib)
CNIEAT_Objbsplibdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_Objbsplibdlib = -x
LEAT_Objbsplibdlib = $(LEAT_exebCountersdlib) $(LLATW16_Objbsplibdlib)
REAT_Objbsplibdlib = $(REAT_exebCountersdlib) $(RLATW16_Objbsplibdlib)
BEAT_Objbsplibdlib = $(BEAT_exebCountersdlib) $(BLATW16_Objbsplibdlib)

CEAT_Objbsplibdlib = $(CEAT_exebDevelopdexe) $(CLATW16_Objbsplibdlib)
CNIEAT_Objbsplibdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;WIN31;Use_Counters;Use_ImageCCD;Use_Library
LNIEAT_Objbsplibdlib = -x
LEAT_Objbsplibdlib = $(LEAT_exebDevelopdexe) $(LLATW16_Objbsplibdlib)
REAT_Objbsplibdlib = $(REAT_exebDevelopdexe) $(RLATW16_Objbsplibdlib)
BEAT_Objbsplibdlib = $(BEAT_exebDevelopdexe) $(BLATW16_Objbsplibdlib)

############################################################################
#
#mehrfach (nicht benutzt!)
#
CLATW16_Objbmotorsdlib =
LLATW16_Objbmotorsdlib =
RLATW16_Objbmotorsdlib =
BLATW16_Objbmotorsdlib =
```

```
CEAT_Objbmotorsdlib = $(CEAT_exebCountersdlib) $(CLATW16_Objbmotorsdlib)
CNIEAT_Objbmotorsdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;Build_Counters;Use_Library
LNIEAT_Objbmotorsdlib = -x
LEAT_Objbmotorsdlib = $(LEAT_exebCountersdlib) $(LLATW16_Objbmotorsdlib)
REAT_Objbmotorsdlib = $(REAT_exebCountersdlib) $(RLATW16_Objbmotorsdlib)
BEAT_Objbmotorsdlib = $(BEAT_exebCountersdlib) $(BLATW16_Objbmotorsdlib)

CEAT_Objbmotorsdlib = $(CEAT_exebDevelopdexe) $(CLATW16_Objbmotorsdlib)
CNIEAT_Objbmotorsdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;WIN31;Use_Counters;Use_ImageCCD;Use_Library
LNIEAT_Objbmotorsdlib = -x
LEAT_Objbmotorsdlib = $(LEAT_exebDevelopdexe) $(LLATW16_Objbmotorsdlib)
REAT_Objbmotorsdlib = $(REAT_exebDevelopdexe) $(RLATW16_Objbmotorsdlib)
BEAT_Objbmotorsdlib = $(BEAT_exebDevelopdexe) $(BLATW16_Objbmotorsdlib)
#
############################################################################

CLATW16_Objbcountersdlib =
LLATW16_Objbcountersdlib =
RLATW16_Objbcountersdlib =
BLATW16_Objbcountersdlib =
CEAT_Objbcountersdlib = $(CEAT_exebDevelopdexe) $(CLATW16_Objbcountersdlib)
CNIEAT_Objbcountersdlib = -I$(INCLUDE) -D_RTLDLL;_BIDSDLL;WIN31;Use_Counters;Use_ImageCCD;Use_Library
LNIEAT_Objbcountersdlib = -x
LEAT_Objbcountersdlib = $(LEAT_exebDevelopdexe) $(LLATW16_Objbcountersdlib)
REAT_Objbcountersdlib = $(REAT_exebDevelopdexe) $(RLATW16_Objbcountersdlib)
BEAT_Objbcountersdlib = $(BEAT_exebDevelopdexe) $(BLATW16_Objbcountersdlib)



############################################################################
#
# Targets
#
Dep_xcontrol = \
   obj\SPLib.lib\
   obj\Motors.lib\
   obj\Counters.lib\
   exe\Develop.exe\
   exe\sphelp.hlp

Dep_xcontrol_ctc = \
   $(CTCDIR)\obj\SPLib.lib\
   $(CTCDIR)\obj\Motors.lib\
   $(CTCDIR)\obj\Counters.lib\
   exe\cDevelop.exe\
   exe\sphelp.hlp

xcontrol : objectdir BccW16.cfg $(Dep_xcontrol)
  @echo ... nicht instrumentiertes Programm im Verzeichnis
  @echo $(XCTLHOME) fertig gestellt !

ctc : ctcinstrument ctccopy BccW16.cfg $(Dep_xcontrol_ctc)
  @echo ... istrumentiertes Programm im Verzeichnis
  @echo $(XCTLHOME) fertig gestellt !



############################################################################
#
# Develop.exe
#
Dep_exebDevelopdexe = \
```

```
   .\OBJ\m_dlgdif.obj\
   .\OBJ\dlg_tpl.obj\
   .\OBJ\m_dlg.obj\
   .\OBJ\m_justag.obj\
   .\OBJ\transfrm.obj\
   .\OBJ\matrix.obj\
   .\OBJ\m_data.obj\
   .\OBJ\m_arscan.obj\
   .\OBJ\m_scan.obj\
   .\OBJ\m_ccdscn.obj\
   .\OBJ\m_steerg.obj\
   .\OBJ\m_main.obj\
   .\OBJ\m_topo.obj\
   .\OBJ\m_device.obj\
   .\OBJ\msimstat.obj\
   .\OBJ\main.res\
   internls\main.def\
   Obj\splib.lib\
   Obj\motors.lib\
   Obj\counters.lib

Dep_exebDevelopdexe_ctc = \
   .\$(CTCDIR)\OBJ\m_dlgdif.obj\
   .\$(CTCDIR)\OBJ\dlg_tpl.obj\
   .\$(CTCDIR)\OBJ\m_dlg.obj\
   .\$(CTCDIR)\OBJ\m_justag.obj\
   .\$(CTCDIR)\OBJ\transfrm.obj\
   .\$(CTCDIR)\OBJ\matrix.obj\
   .\$(CTCDIR)\OBJ\m_data.obj\
   .\$(CTCDIR)\OBJ\m_arscan.obj\
   .\$(CTCDIR)\OBJ\m_scan.obj\
   .\$(CTCDIR)\OBJ\m_ccdscn.obj\
   .\$(CTCDIR)\OBJ\m_steerg.obj\
   .\$(CTCDIR)\OBJ\m_main.obj\
   .\$(CTCDIR)\OBJ\m_topo.obj\
   .\$(CTCDIR)\OBJ\m_device.obj\
   .\$(CTCDIR)\OBJ\msimstat.obj\
   .\$(CTCDIR)\OBJ\mon.obj\
   .\OBJ\main.res\
   internls\main.def\
   $(CTCDIR)\obj\splib.lib\
   $(CTCDIR)\obj\motors.lib\
   $(CTCDIR)\obj\counters.lib

exe\Develop.exe : $(Dep_exebDevelopdexe)
  $(TLINK)  @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebDevelopdexe) $(LNIEAT_exebDevelopdexe) +
.\BC4LIBS\c0wl.obj+
.\OBJ\m_dlgdif.obj+
.\OBJ\dlg_tpl.obj+
.\OBJ\m_dlg.obj+
.\OBJ\m_justag.obj+
.\OBJ\transfrm.obj+
.\OBJ\matrix.obj+
.\OBJ\m_data.obj+
.\OBJ\m_arscan.obj+
.\OBJ\m_scan.obj+
.\OBJ\m_ccdscn.obj+
.\OBJ\m_steerg.obj+
.\OBJ\m_main.obj+
.\OBJ\m_topo.obj+
.\OBJ\m_device.obj+
```

```
.\OBJ\msimstat.obj
$<,$*
obj\splib.lib+
obj\motors.lib+
obj\counters.lib+
.\BC4LIBS\bwcc.lib+
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib
internls\main.def
|
   $(BRC) .\OBJ\main.res $<

exe\cDevelop.exe : $(Dep_exebDevelopdexe_ctc)
  $(TLINK) @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebDevelopdexe) $(LNIEAT_exebDevelopdexe) +
.\BC4LIBS\c0wl.obj+
.\$(CTCDIR)\OBJ\m_dlgdif.obj+
.\$(CTCDIR)\OBJ\dlg_tpl.obj+
.\$(CTCDIR)\OBJ\m_dlg.obj+
.\$(CTCDIR)\OBJ\m_justag.obj+
.\$(CTCDIR)\OBJ\transfrm.obj+
.\$(CTCDIR)\OBJ\matrix.obj+
.\$(CTCDIR)\OBJ\m_data.obj+
.\$(CTCDIR)\OBJ\m_arscan.obj+
.\$(CTCDIR)\OBJ\m_scan.obj+
.\$(CTCDIR)\OBJ\m_ccdscn.obj+
.\$(CTCDIR)\OBJ\m_steerg.obj+
.\$(CTCDIR)\OBJ\m_main.obj+
.\$(CTCDIR)\OBJ\m_topo.obj+
.\$(CTCDIR)\OBJ\m_device.obj+
.\$(CTCDIR)\OBJ\msimstat.obj+
.\$(CTCDIR)\OBJ\mon.obj
exe\develop.exe,exe\develop
$(CTCDIR)\obj\splib.lib+
$(CTCDIR)\obj\motors.lib+
$(CTCDIR)\obj\counters.lib+
.\BC4LIBS\bwcc.lib+
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib+
$(CTCHOME)\libbc31\ctcwinl.lib
internls\main.def
|
   $(BRC) .\OBJ\main.res exe\develop.exe



#############################################################################
#
# SPLib.dll
#
obj\SPLib.lib : exe\SPLib.dll
  $(IMPLIB) $@ exe\SPLib.dll

$(CTCDIR)\obj\SPLib.lib : exe\cSPLib.dll
  $(IMPLIB) $@ exe\SPLib.dll

Dep_exebSPLibddll = \
   .\OBJ\dlg_tpl.obj\
   .\OBJ\l_layer.obj\
   .\OBJ\m_curve.obj\
   .\OBJ\splib.res\
   internls\splib.def
```

```
Dep_exebSPLibddll_ctc = \
   .\$(CTCDIR)\OBJ\dlg_tpl.obj\
   .\$(CTCDIR)\OBJ\l_layer.obj\
   .\$(CTCDIR)\OBJ\m_curve.obj\
   .\$(CTCDIR)\OBJ\mon.obj\
   .\OBJ\splib.res\
   internls\splib.def

exe\SPLib.dll : $(Dep_exebSPLibddll)
  $(TLINK)   @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebSPLibdlib) $(LNIEAT_exebSPLibdlib) +
.\BC4LIBS\c0dl.obj+
.\OBJ\dlg_tpl.obj+
.\OBJ\l_layer.obj+
.\OBJ\m_curve.obj
$<,$*
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib
internls\splib.def
|
   $(BRC) .\OBJ\splib.res $<

exe\cSPLib.dll : $(Dep_exebSPLibddll_ctc)
  $(TLINK) @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebSPLibdlib) $(LNIEAT_exebSPLibdlib) +
.\BC4LIBS\c0dl.obj+
.\$(CTCDIR)\OBJ\dlg_tpl.obj+
.\$(CTCDIR)\OBJ\l_layer.obj+
.\$(CTCDIR)\OBJ\m_curve.obj+
.\$(CTCDIR)\OBJ\mon.obj
exe\splib.dll,exe\splib
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib+
$(CTCHOME)\libbc31\ctcwdll.lib
internls\splib.def
|
   $(BRC) .\OBJ\splib.res exe\splib.dll



#############################################################################
#
# Motors.dll
#
obj\Motors.lib : exe\Motors.dll
  $(IMPLIB) $@ exe\Motors.dll

$(CTCDIR)\obj\Motors.lib : exe\cMotors.dll
  $(IMPLIB) $@ exe\Motors.dll

Dep_exebMotorsddll = \
   .\OBJ\dlg_tpl.obj\
   .\OBJ\motors.obj\
   .\OBJ\m_layer.obj\
   .\OBJ\motors.res\
   motrstrg\motors.def\
   obj\splib.lib

Dep_exebMotorsddll_ctc = \
   .\$(CTCDIR)\OBJ\dlg_tpl.obj\
   .\$(CTCDIR)\OBJ\motors.obj\
```

```
   .\$(CTCDIR)\OBJ\m_layer.obj\
   .\$(CTCDIR)\OBJ\mon.obj\
   .\OBJ\motors.res\
   motrstrg\motors.def\
   $(CTCDIR)\obj\splib.lib

exe\Motors.dll : $(Dep_exebMotorsddll)
  $(TLINK)   @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebMotorsdlib) $(LNIEAT_exebMotorsdlib) +
.\BC4LIBS\c0dl.obj+
.\OBJ\dlg_tpl.obj+
.\OBJ\motors.obj+
.\OBJ\m_layer.obj
$<,$*
obj\splib.lib+
.\BC4LIBS\bwcc.lib+
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib
motrstrg\motors.def
|
   $(BRC) .\OBJ\motors.res $<

exe\cMotors.dll : $(Dep_exebMotorsddll_ctc)
  $(TLINK) @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebMotorsdlib) $(LNIEAT_exebMotorsdlib) +
.\BC4LIBS\c0dl.obj+
.\$(CTCDIR)\OBJ\dlg_tpl.obj+
.\$(CTCDIR)\OBJ\motors.obj+
.\$(CTCDIR)\OBJ\m_layer.obj+
.\$(CTCDIR)\OBJ\mon.obj
exe\motors.dll,exe\motors
$(CTCDIR)\obj\splib.lib+
.\BC4LIBS\bwcc.lib+
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib+
$(CTCHOME)\libbc31\ctcwdll.lib
motrstrg\motors.def
|
   $(BRC) .\OBJ\motors.res exe\motors.dll



#############################################################################
#
# Counters.dll
#
obj\Counters.lib : exe\Counters.dll
  $(IMPLIB) $@ exe\Counters.dll

$(CTCDIR)\obj\Counters.lib : exe\cCounter.dll
  $(IMPLIB) $@ exe\Counters.dll

Dep_exebCountersddll = \
   .\OBJ\dlg_tpl.obj\
   .\OBJ\testdev.obj\
   .\OBJ\counters.obj\
   .\OBJ\c_layer.obj\
   .\OBJ\am9513.obj\
   .\OBJ\braunpsd.obj\
   .\OBJ\kisl1.obj\
   .\OBJ\kmpt1.obj\
   .\OBJ\counters.res\
```

```
   detecuse\counters.def\
   obj\motors.lib\
   obj\splib.lib

Dep_exebCountersddll_ctc = \
   .\$(CTCDIR)\OBJ\dlg_tpl.obj\
   .\$(CTCDIR)\OBJ\testdev.obj\
   .\$(CTCDIR)\OBJ\counters.obj\
   .\$(CTCDIR)\OBJ\c_layer.obj\
   .\$(CTCDIR)\OBJ\am9513.obj\
   .\$(CTCDIR)\OBJ\braunpsd.obj\
   .\$(CTCDIR)\OBJ\kisl1.obj\
   .\$(CTCDIR)\OBJ\kmpt1.obj\
   .\$(CTCDIR)\OBJ\mon.obj\
   .\OBJ\counters.res\
   detecuse\counters.def\
   $(CTCDIR)\obj\motors.lib\
   $(CTCDIR)\obj\splib.lib

exe\Counters.dll : $(Dep_exebCountersddll)
  $(TLINK)   @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebCountersdlib) $(LNIEAT_exebCountersdlib) +
.\BC4LIBS\c0dl.obj+
.\OBJ\dlg_tpl.obj+
.\OBJ\testdev.obj+
.\OBJ\counters.obj+
.\OBJ\c_layer.obj+
.\OBJ\am9513.obj+
.\OBJ\braunpsd.obj+
.\OBJ\kisl1.obj+
.\OBJ\kmpt1.obj
$<,$*
obj\motors.lib+
obj\splib.lib+
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib
detecuse\counters.def
|
   $(BRC) .\OBJ\counters.res $<

exe\cCounter.dll : $(Dep_exebCountersddll_ctc)
  $(TLINK) @&&|
 /v $(IDE_LFLAGS) $(LEAT_exebCountersdlib) $(LNIEAT_exebCountersdlib) +
.\BC4LIBS\c0dl.obj+
.\$(CTCDIR)\OBJ\dlg_tpl.obj+
.\$(CTCDIR)\OBJ\testdev.obj+
.\$(CTCDIR)\OBJ\counters.obj+
.\$(CTCDIR)\OBJ\c_layer.obj+
.\$(CTCDIR)\OBJ\am9513.obj+
.\$(CTCDIR)\OBJ\braunpsd.obj+
.\$(CTCDIR)\OBJ\kisl1.obj+
.\$(CTCDIR)\OBJ\kmpt1.obj+
.\$(CTCDIR)\OBJ\mon.obj
exe\counters.dll,exe\counters
$(CTCDIR)\obj\motors.lib+
$(CTCDIR)\obj\splib.lib+
.\BC4LIBS\import.lib+
.\BC4LIBS\crtldll.lib+
$(CTCHOME)\libbc31\ctcwdll.lib
detecuse\counters.def
|
   $(BRC) .\OBJ\counters.res exe\counters.dll
```

```
#############################################################################
#
# Objektfiles
#
.\OBJ\dlg_tpl.obj :  swintrac\dlg_tpl.cpp
  $(BCC)   -c @&&|
 $(CEAT_swintracbdlg_tpldcpp) $(CNIEAT_swintracbdlg_tpldcpp) -o$@ swintrac\dlg_tpl.cpp
|

.\$(CTCDIR)\OBJ\dlg_tpl.obj :  $(CTCDIR)\dlg_tpl.cpp
  $(BCC)   -c @&&|
 $(CEAT_swintracbdlg_tpldcpp) $(CNIEAT_swintracbdlg_tpldcpp) -o$@ $(CTCDIR)\dlg_tpl.cpp
|

.\OBJ\l_layer.obj :  internls\l_layer.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib) -o$@ internls\l_layer.cpp
|

.\$(CTCDIR)\OBJ\l_layer.obj :  $(CTCDIR)\l_layer.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib) -o$@ $(CTCDIR)\l_layer.cpp
|

.\OBJ\m_curve.obj :  datavisa\m_curve.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib) -o$@ datavisa\m_curve.cpp
|

.\$(CTCDIR)\OBJ\m_curve.obj :  $(CTCDIR)\m_curve.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib) -o$@ $(CTCDIR)\m_curve.cpp
|

.\OBJ\motors.obj :  motrstrg\motors.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebMotorsdlib) $(CNIEAT_exebMotorsdlib) -o$@ motrstrg\motors.cpp
|

.\$(CTCDIR)\OBJ\motors.obj :  $(CTCDIR)\motors.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebMotorsdlib) $(CNIEAT_exebMotorsdlib) -o$@ $(CTCDIR)\motors.cpp
|

.\OBJ\m_layer.obj :  motrstrg\m_layer.cpp
  $(BCC)   -c @&&|
 $(CEAT_motrstrgbm_layerdcpp) $(CNIEAT_motrstrgbm_layerdcpp) -o$@ motrstrg\m_layer.cpp
|

.\$(CTCDIR)\OBJ\m_layer.obj :  $(CTCDIR)\m_layer.cpp
  $(BCC)   -c @&&|
 $(CEAT_motrstrgbm_layerdcpp) $(CNIEAT_motrstrgbm_layerdcpp) -o$@ $(CTCDIR)\m_layer.cpp
|

.\OBJ\testdev.obj :  detecuse\testdev.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -o$@ detecuse\testdev.cpp
|
```

```
.\$(CTCDIR)\OBJ\testdev.obj :  $(CTCDIR)\testdev.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -o$@ $(CTCDIR)\testdev.cpp
|

.\OBJ\counters.obj :  detecuse\counters.cpp
  $(BCC)   -c @&&|
 $(CEAT_detecusebcountersdcpp) $(CNIEAT_detecusebcountersdcpp) -o$@ detecuse\counters.cpp
|

.\$(CTCDIR)\OBJ\counters.obj :  $(CTCDIR)\counters.cpp
  $(BCC)   -c @&&|
 $(CEAT_detecusebcountersdcpp) $(CNIEAT_detecusebcountersdcpp) -o$@ $(CTCDIR)\counters.cpp
|

.\OBJ\c_layer.obj :  detecuse\c_layer.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -o$@ detecuse\c_layer.cpp
|

.\$(CTCDIR)\OBJ\c_layer.obj :  $(CTCDIR)\c_layer.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -o$@ $(CTCDIR)\c_layer.cpp
|

.\OBJ\am9513.obj :  detecuse\am9513.cpp
  $(BCC)   -c @&&|
 $(CEAT_detecusebam9513dcpp) $(CNIEAT_detecusebam9513dcpp) -o$@ detecuse\am9513.cpp
|

.\$(CTCDIR)\OBJ\am9513.obj :  $(CTCDIR)\am9513.cpp
  $(BCC)   -c @&&|
 $(CEAT_detecusebam9513dcpp) $(CNIEAT_detecusebam9513dcpp) -o$@ $(CTCDIR)\am9513.cpp
|

.\OBJ\braunpsd.obj :  detecuse\braunpsd.cpp
  $(BCC)   -c @&&|
 $(CEAT_detecusebbraunpsddcpp) $(CNIEAT_detecusebbraunpsddcpp) -o$@ detecuse\braunpsd.cpp
|

.\$(CTCDIR)\OBJ\braunpsd.obj :  $(CTCDIR)\braunpsd.cpp
  $(BCC)   -c @&&|
 $(CEAT_detecusebbraunpsddcpp) $(CNIEAT_detecusebbraunpsddcpp) -o$@ $(CTCDIR)\braunpsd.cpp
|

.\OBJ\kisl1.obj :  detecuse\kisl1.c
  $(BCC)   -P- -c @&&|
 $(CEAT_detecusebkisl1dc) $(CNIEAT_detecusebkisl1dc) -o$@ detecuse\kisl1.c
|

.\$(CTCDIR)\OBJ\kisl1.obj :  $(CTCDIR)\kisl1.c
  $(BCC)   -P- -c @&&|
 $(CEAT_detecusebkisl1dc) $(CNIEAT_detecusebkisl1dc) -o$@ $(CTCDIR)\kisl1.c
|

.\OBJ\kmpt1.obj :  detecuse\kmpt1.c
  $(BCC)   -P- -c @&&|
 $(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -o$@ detecuse\kmpt1.c
|

.\$(CTCDIR)\OBJ\kmpt1.obj :  $(CTCDIR)\kmpt1.c
  $(BCC)   -P- -c @&&|
```

```
 $(CEAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -o$@ $(CTCDIR)\kmpt1.c
|

.\OBJ\m_dlg.obj :  swintrac\m_dlg.cpp
  $(BCC)   -c @&&|
 $(CEAT_swintracbm_dlgdcpp) $(CNIEAT_swintracbm_dlgdcpp) -o$@ swintrac\m_dlg.cpp
|

.\$(CTCDIR)\OBJ\m_dlg.obj :  $(CTCDIR)\m_dlg.cpp
  $(BCC)   -c @&&|
 $(CEAT_swintracbm_dlgdcpp) $(CNIEAT_swintracbm_dlgdcpp) -o$@ $(CTCDIR)\m_dlg.cpp
|

.\OBJ\m_justag.obj :  autojust\m_justag.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ autojust\m_justag.cpp
|

.\$(CTCDIR)\OBJ\m_justag.obj :  $(CTCDIR)\m_justag.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_justag.cpp
|

.\OBJ\transfrm.obj :  autojust\transfrm.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ autojust\transfrm.cpp
|

.\$(CTCDIR)\OBJ\transfrm.obj :  $(CTCDIR)\transfrm.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\transfrm.cpp
|

.\OBJ\matrix.obj :  autojust\matrix.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ autojust\matrix.cpp
|

.\$(CTCDIR)\OBJ\matrix.obj :  $(CTCDIR)\matrix.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\matrix.cpp
|

.\OBJ\m_data.obj :  datavisa\m_data.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ datavisa\m_data.cpp
|

.\$(CTCDIR)\OBJ\m_data.obj :  $(CTCDIR)\m_data.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_data.cpp
|

.\OBJ\m_arscan.obj :  difrkmty\m_arscan.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ difrkmty\m_arscan.cpp
|

.\$(CTCDIR)\OBJ\m_arscan.obj :  $(CTCDIR)\m_arscan.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_arscan.cpp
|
```

```
.\OBJ\m_scan.obj :  difrkmty\m_scan.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ difrkmty\m_scan.cpp
|

.\$(CTCDIR)\OBJ\m_scan.obj :  $(CTCDIR)\m_scan.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_scan.cpp
|

.\OBJ\m_ccdscn.obj :  difrkmty\m_ccdscn.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ difrkmty\m_ccdscn.cpp
|

.\$(CTCDIR)\OBJ\m_ccdscn.obj :  $(CTCDIR)\m_ccdscn.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_ccdscn.cpp
|

.\OBJ\m_steerg.obj :  workflow\m_steerg.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ workflow\m_steerg.cpp
|

.\$(CTCDIR)\OBJ\m_steerg.obj :  $(CTCDIR)\m_steerg.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_steerg.cpp
|

.\OBJ\m_main.obj :  internls\m_main.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ internls\m_main.cpp
|

.\$(CTCDIR)\OBJ\m_main.obj :  $(CTCDIR)\m_main.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_main.cpp
|

.\OBJ\m_topo.obj :  topogrfy\m_topo.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ topogrfy\m_topo.cpp
|

.\$(CTCDIR)\OBJ\m_topo.obj :  $(CTCDIR)\m_topo.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_topo.cpp
|

.\OBJ\m_device.obj :  swintrac\m_device.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ swintrac\m_device.cpp
|

.\$(CTCDIR)\OBJ\m_device.obj :  $(CTCDIR)\m_device.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_device.cpp
|

.\OBJ\msimstat.obj :  motrstrg\msimstat.cpp
```

```
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ motrstrg\msimstat.cpp
|

.\$(CTCDIR)\OBJ\msimstat.obj :  $(CTCDIR)\msimstat.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\msimstat.cpp
|

.\OBJ\m_dlgdif.obj :  difrkmty\m_dlgdif.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ difrkmty\m_dlgdif.cpp
|

.\$(CTCDIR)\OBJ\m_dlgdif.obj :  $(CTCDIR)\m_dlgdif.cpp
  $(BCC)   -c @&&|
 $(CEAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -o$@ $(CTCDIR)\m_dlgdif.cpp
|

#############################################################################
#
# Monitor wird bei Instrumentierung automatisch erzeugt
#
.\$(CTCDIR)\OBJ\mon.obj :  $(CTCDIR)\mon.cpp
  bcc -ml -w-par -c -o$@ $(CTCDIR)\mon.cpp
#
#############################################################################



#############################################################################
#
# Ressourcen
#
.\OBJ\splib.res :  winresrc\splib.rc
  $(BRC) -R @&&|
 $(IDE_RFLAGS) $(REAT_exebSPLibdlib) $(CNIEAT_exebSPLibdlib) -FO$@ winresrc\splib.rc
|

.\OBJ\motors.res :  winresrc\motors.rc
  $(BRC) -R @&&|
 $(IDE_RFLAGS) $(REAT_exebMotorsdlib) $(CNIEAT_exebMotorsdlib) -FO$@ winresrc\motors.rc
|

.\OBJ\main.res :  winresrc\main.rc
  $(BRC) -R @&&|
 $(IDE_RFLAGS) $(REAT_exebDevelopdexe) $(CNIEAT_exebDevelopdexe) -FO$@ winresrc\main.rc
|

.\OBJ\counters.res :  winresrc\counters.rc
  $(BRC) -R @&&|
 $(IDE_RFLAGS) $(REAT_exebCountersdlib) $(CNIEAT_exebCountersdlib) -FO$@ winresrc\counters.rc
|



#############################################################################
#
# Help
#
Dep_helpbsphelpdhlp = \
   help\sphelp.rtf
```

```
exe\sphelp.hlp : $(Dep_helpbsphelpdhlp)
  $(CompileHLP)   help\sphelp.hpj
  Move sphelp.hlp exe\sphelp.hlp


#----------------------------------------------------------------------------
#
# Cleanup
#
clean:
        @if exist OBJ\nul for %a in (OBJ\*.*) do del %a
        @if exist EXE\Develop.exe del EXE\Develop.exe
        @if exist EXE\Motors.dll del EXE\Motors.dll
        @if exist EXE\Counters.dll del EXE\Counters.dll
        @if exist EXE\SPLib.dll del EXE\SPLib.dll
        @if exist EXE\$(MONITOR).* del EXE\$(MONITOR).*
        @if exist EXE\sphelp.hlp del EXE\sphelp.hlp
        @if exist $(CTCDIR)\OBJ\nul for %a in ($(CTCDIR)\OBJ\*.*) do del %a
        @if exist $(CTCDIR)\OBJ\nul rd $(CTCDIR)\OBJ
        @if exist $(CTCDIR)\nul for %a in ($(CTCDIR)\*.*) do del %a
        @if exist $(CTCDIR)\nul rd $(CTCDIR)
        @if exist *.lst del *.lst
        @if exist CTCHTML\nul for %a in (CTCHTML\*.*) do del %a
        @if exist CTCHTML\nul rd CTCHTML
        @if exist profile.lst del profile.lst
        @if exist summary.lst del summary.lst
        @if exist untested.lst del untested.lst


#----------------------------------------------------------------------------
#
# Compiler configuration file
#
BccW16.cfg :
   Copy &&|
-R
-v
-vi
-H
-y
-N
-vi-
-4
-a
-Fs-
-dc
-Vf
-h-
-Vmp
-Vmv
-Vb
-xp
-xf
-xc-
-Oc
-Ot
-O-a
-Om
-Op
```

```
-Oi
-Ov
-wcln
-w-sig
-wucp
-wbig
-wbbf
-wpin
-wnak
-wpre
-wdef
-wnod
-wamb
-wuse
-wstv
-wasm
-wamp
-wobs
-wpch
-w-aus
-d
-po
-Vt-
-Vv-
-K
-Vc-
-Vp-
-H""
-W
-Va
-Z
-O
-Ol
-Ob
-OW
-Ff
-Ff=1000
-rd
-ml
-g200
-j10
-H
-Od
-v
| $@
```

E.2 CTC++ Protokolle

Die folgenden Protokolle entstanden nach Ausführen aller definierten Testfälle auf einer instrumentierten XCTL-Version vom 20. Januar 2002. Vor jeder von CTC++ registrierten Funktion stehen drei Zahlen. Dabei wir die Häufigkeit der Funktionsaufrufe durch die erste Zahl wiedergegeben. Ob das Ende einer Funktion erreicht wurde, ist der zweiten Zahl zu entnehmen. Funtionen, die vorzeitig durch ein *return* abgebrochen werden, führen daher nicht zur Erhöhung dieses Wertes. Die dritte Zahl gibt lediglich die Zeilennummer der Funktion innerhalb der C bzw. C++ Datei an.

Dabei ist zu beachten, dass die Werte nach Übersteigen von 999, in exponentierter Form dargestellt werden. Beträgt die tatsächliche Anzahl der Funktionsaufrufe `4376`, wird in der Tabelle lediglich der Wert `4E3` dargestellt. Die genauen Werte können bei Bedarf aus den angelegten ASCII-Reportdateien `profile.lst`, `untested.lst` bzw. `summary.lst` entnommen werden. Für unsere Untersuchungen, ob die Funktionen mindestens einmal aufgerufen wurden, ist diese Darstellung völlig ausreichend.

```
MONITORED SOURCE FILE : AUTOJUST\M_JUSTAG.CPP

#line 1 ".\INCLUDE\autojust\matrix.h"
  0    0   60 FUNCTION TMatrix::ist_homogen()
  0    0  118 FUNCTION TVektor::TVektor()
  0    0  125 FUNCTION TVektor::TVektor()
  0    0  130 FUNCTION TVektor::TVektor()
  2    2  193 FUNCTION TMatrizenListe::~TMatrizenListe()
  0    0  201 FUNCTION TMatrizenListe::ist_leer()
  0    0  207 FUNCTION TMatrizenListe::elementanzahl()
#line 1 ".\INCLUDE\autojust\transfrm.h"
1E3    0  111 FUNCTION TransformationClass::GetOrigPosBorders()
  0    0  123 FUNCTION TransformationClass::get_koordinatentrafos()
#line 23 "autojust\m_justag.cpp"
3E3 3E3   36 FUNCTION WriteToJustageLog()
  1    1   62 FUNCTION TAutomaticAngleControl::TAutomaticAngleControl()
  1    1   67 try
  1    0   82 FUNCTION TAutomaticAngleControl::Dlg_OnInit()
128 128  173 FUNCTION TAutomaticAngleControl::Dlg_OnCommand()
  1    0  844 FUNCTION TAutomaticAngleControl::LeaveDialog()
*** TER  53%  (  8/ 15) of SOURCE FILE AUTOJUST\M_JUSTAG.CPP

MONITORED SOURCE FILE : AUTOJUST\MATRIX.CPP

#line 1 ".\INCLUDE\autojust\matrix.h"
1E3    0   60 FUNCTION TMatrix::ist_homogen()
  0    0  118 FUNCTION TVektor::TVektor()
  0    0  125 FUNCTION TVektor::TVektor()
  0    0  130 FUNCTION TVektor::TVektor()
  0    0  193 FUNCTION TMatrizenListe::~TMatrizenListe()
  0    0  201 FUNCTION TMatrizenListe::ist_leer()
  0    0  207 FUNCTION TMatrizenListe::elementanzahl()
#line 5 "autojust\matrix.cpp"
6E3 6E3   13 FUNCTION TMatrix::TMatrix()
4E3 4E3   21 FUNCTION TMatrix::TMatrix()
2E4 2E4   30 FUNCTION TMatrix::~TMatrix()
1E4 1E4   36 FUNCTION TMatrix::TMatrix()
6E3    0  47 FUNCTION TMatrix::operator_=()
  0    0   61 FUNCTION TMatrix::operator_+()
  0    0   78 FUNCTION TMatrix::operator_-()
  0    0   96 FUNCTION TMatrix::operator_*()
  0    0  104 FUNCTION operator_*()
3E3    0  114 FUNCTION TMatrix::operator_*()
112    0  135 FUNCTION TMatrix::einheitsmatrix()
  0    0  157 FUNCTION TMatrix::invers()
 28    0  198 FUNCTION TMatrix::verschiebematrix()
 28    0  223 FUNCTION TMatrix::rotationsmatrix_x()
 28    0  238 FUNCTION TMatrix::rotationsmatrix_y()
 28    0  253 FUNCTION TMatrix::rotationsmatrix_z()
 28    0  269 FUNCTION TMatrix::transformiere()
3E3 3E3  321 FUNCTION TVektor::TVektor()
 14    0  340 FUNCTION operator_*()
533    0  349 FUNCTION TVektor::mache_homogen()
533    0  381 FUNCTION TVektor::mache_kartesisch()
 28    0  411 FUNCTION TVektor::vektor_betrag()
 14    0  435 FUNCTION TVektor::skalarprodukt()
 14    0  469 FUNCTION TVektor::winkel()
940    0  504 FUNCTION TVektor::set_XYZ()
1E3    0  515 FUNCTION TVektor::get_XYZ()
  2    2  536 FUNCTION TMatrizenListe::TMatrizenListe()
 28    0  549 FUNCTION TMatrizenListe::push()
  0    0  561 FUNCTION TMatrizenListe::pop()
3E3    0  572 FUNCTION TMatrizenListe::zeige()
*** TER  68%  ( 25/ 37) of SOURCE FILE AUTOJUST\MATRIX.CPP
```

MONITORED SOURCE FILE : AUTOJUST\TRANSFRM.CPP

```
#line 1 ".\INCLUDE\autojust\matrix.h"
  0    0   60 FUNCTION TMatrix::ist_homogen()
  1    1  118 FUNCTION TVektor::TVektor()
1E3  1E3  125 FUNCTION TVektor::TVektor()
  1    1  130 FUNCTION TVektor::TVektor()
  0    0  193 FUNCTION TMatrizenListe::~TMatrizenListe()
  0    0  201 FUNCTION TMatrizenListe::ist_leer()
1E3    0  207 FUNCTION TMatrizenListe::elementanzahl()
#line 1 ".\INCLUDE\autojust\transfrm.h"
  0    0  111 FUNCTION TransformationClass::GetOrigPosBorders()
421    0  123 FUNCTION TransformationClass::get_koordinatentrafos()
#line 13 "autojust\transfrm.cpp"
  1    1   30 FUNCTION TransformationClass::TransformationClass()
  1    1   37 try
  1    1   53 FUNCTION TransformationClass::~TransformationClass()
  1    0   60 FUNCTION TransformationClass::Initialize()
 98    0  112 FUNCTION TransformationClass::translate_from_worldpoints()
435    0  130 FUNCTION TransformationClass::translate_to_worldpoints()
 14    0  149 FUNCTION TransformationClass::translate_PosBorders()
421    0  190 FUNCTION TransformationClass::GetIntensityOnPosition()
 28    0  252 FUNCTION TransformationClass::Goldener_Schnitt()
448    0  347 FUNCTION TransformationClass::MeasureIntensity()
448  448  353 try
 14    0  468 FUNCTION TransformationClass::KoordinatenTransformation()
  1    1  523 FUNCTION TransformationClass::DFCorrection()
*** TER  82%  ( 18/ 22) of SOURCE FILE AUTOJUST\TRANSFRM.CPP
```

MONITORED SOURCE FILE : DATAVISA\M_CURVE.CPP

```
 23   23   30 FUNCTION TDataBase::TDataBase()
  0    0   42 FUNCTION TDataBase::~TDataBase()
 23    0   53 FUNCTION TDataBase::New()
1E3    0   70 FUNCTION TDataBase::AddCurve()
  2    0   92 FUNCTION TDataBase::DelCurve()
  6    6  104 FUNCTION TDataBase::SetCIdx()
1E4    0  111 FUNCTION TDataBase::GetCurve()
2E3  2E3  136 FUNCTION TCurve::TCurve()
2E3  2E3  183 FUNCTION TCurve::~TCurve()
2E3    0  198 FUNCTION TCurve::New()
  0    0  223 FUNCTION TCurve::BackStep()
  4    0  232 FUNCTION TCurve::operator_=()
3E5    0  282 FUNCTION TCurve::FastPAdd()
9E5    0  330 FUNCTION TCurve::FastPGet()
1E4    0  367 FUNCTION TCurve::FastOpen()
1E4    0  424 FUNCTION TCurve::FastClose()
511    0  441 FUNCTION TCurve::PAdd()
6E3    0  485 FUNCTION TCurve::PGet()
203    0  538 FUNCTION TCurve::GetValueByValue()
1E4    0  585 FUNCTION TCurve::SetPP()
  0    0  604 FUNCTION TCurve::ValueAdd()
  0    0  640 FUNCTION TCurve::Save()
  0    0  679 FUNCTION TCurve::Save()
2E3    0  729 FUNCTION TCurve::GetGravityCenter()
  0    0  801 FUNCTION TCurve::DeleteUnderGround()
  0    0  833 FUNCTION TCurve::DeleteFlanks()
  0    0  992 FUNCTION TCurve::GetPeakProperties()
*** TER  70%  ( 19/ 27) of SOURCE FILE DATAVISA\M_CURVE.CPP
```

MONITORED SOURCE FILE : DATAVISA\M_DATA.CPP

```
  2   2   89 FUNCTION TBitmapSource::TBitmapSource()
  3   3  134 FUNCTION TBitmapSource::BereichDef()
  4   4  151 FUNCTION TBitmapSource::ColorDef()
  2   2  202 FUNCTION TBitmapSource::~TBitmapSource()
  0   0  218 FUNCTION TBitmapSource::New()
  0   0  239 FUNCTION TBitmapSource::ProcessBitmapFile()
 12   0  304 FUNCTION TBitmapSource::GetImageSize()
  6   0  322 FUNCTION TBitmapSource::CreateDefaultPalette()
 34   0  405 FUNCTION TBitmapSource::CreatePaletteFromDIB()
  6   6  445 FUNCTION TBitmapSource::FillBMInfoFromPalette()
 34   0  468 FUNCTION TBitmapSource::GetPalette()
  6   0  480 FUNCTION TBitmapSource::CreateGDIObject()
7E5   0  500 FUNCTION TBitmapSource::GetColor()
 24  24  549 FUNCTION TBitmapSource::DrawBitmap()
 24  24  573 FUNCTION TBitmapSource::DrawBitmapFrame()
 48  48  599 FUNCTION TBitmapSource::SetScreen()
 24  24  634 FUNCTION TBitmapSource::DrawColorTable()
  4   4  782 FUNCTION TBitmapSource::GenerateAngleSpaceBitmap()
 24  24  980 FUNCTION TBitmapSource::DrawMeasurementArea()
  2   2 1144 FUNCTION TBitmapSource::GenerateRLBitmap()
  1   1 1292   try
  6   6 1431 FUNCTION TBitmapSource::FormatDBaseToBitmapSource()
  0   0 1549 FUNCTION TBitmapSource::RenderDIB()
  1   1 1585 FUNCTION TBitmapSource::UpdateBitmapSource()
 12  12 1593 FUNCTION TPlotData::TPlotData()
 12  12 1670 FUNCTION TPlotData::~TPlotData()
 10   0 1680 FUNCTION TPlotData::New()
356 356 1720 FUNCTION TPlotData::SetRanges()
  1   1 1744 FUNCTION TPlotData::lButtonDown()
187  38 1949 FUNCTION TPlotData::MouseMove()
  1   1 2032 FUNCTION TPlotData::lButtonUp()
  0   0 2041 FUNCTION TPlotData::SetMeasurementArea()
  0   0 2058 FUNCTION TPlotData::RenderFormat()
  0   0 2086 FUNCTION TPlotData::RenderAllFormats()
  0   0 2099 FUNCTION TPlotData::DoCopy()
248 248 2122 FUNCTION TPlotData::DrawCoordinateSystem()
2E3   0 2361 FUNCTION TPlotData::GetNachkommaStelle()
557   0 2388 FUNCTION TPlotData::Paint()
 14  14 2559 FUNCTION TPlotData::SetFileType()
460 460 2566 FUNCTION TPlotData::FormatCurveToPLine()
  2   2 2637 FUNCTION TPlotData::SetKSProperties()
  2   2 2656 FUNCTION TPlotData::PickUpData()
  2   2 2667 FUNCTION TPlotData::KillSecondCurve()
  2   2 2678 FUNCTION TPlotData::FreezeCurve()
  4   0 2690 FUNCTION TPlotData::SaveSecondCurve()
  1   1 2768 FUNCTION TCurveFreeScal::TCurveFreeScal()
  1   0 2795 FUNCTION TCurveFreeScal::Dlg_OnInit()
124 124 2827 FUNCTION TCurveFreeScal::Dlg_OnCommand()
  2   0 2876 FUNCTION TCurveFreeScal::CanClose()
  1   0 2998 FUNCTION TCurveFreeScal::LeaveDialog()
  3   3 3006 FUNCTION TCurveShowParam::TCurveShowParam()
  3   0 3016 FUNCTION TCurveShowParam::Dlg_OnInit()
  0   0 3143 FUNCTION TCurveShowParam::Dlg_OnHScrollBar()
 95  95 3227 FUNCTION TCurveShowParam::Dlg_OnCommand()
  8   0 3375 FUNCTION TCurveShowParam::CanClose()
  3   0 3585 FUNCTION TCurveShowParam::LeaveDialog()
  1   1 3600 FUNCTION TCurveFreeScalColor::TCurveFreeScalColor()
  1   1 3609 FUNCTION TCurveFreeScalColor::~TCurveFreeScalColor()
  3   0 3624 FUNCTION TCurveFreeScalColor::SetTitle()
  1   1 3637 FUNCTION TCurveFreeScalColor::Create()
  4   4 3650 FUNCTION TCurveFreeScalColor::Paint()
  4   4 3656 FUNCTION TCurveFreeScalColor::DrawColorTable()
*** TER  87%  ( 54/ 62) of SOURCE FILE DATAVISA\M_DATA.CPP
```

MONITORED SOURCE FILE : DETECUSE\AM9513.CPP

```
  0    0   10 FUNCTION TAm9513a::TAm9513a()
  0    0   21 FUNCTION TAm9513a::LoockUp()
  0    0   31 FUNCTION TAm9513a::Init()
  0    0   70 FUNCTION TAm9513a::IOCTL()
  0    0   76 FUNCTION TAm9513a::IOCTL()
  0    0  218 FUNCTION TAm9513a::SplitNumber()
  0    0  278 FUNCTION TAm9513a::GetTicksPerSecond()
  0    0  283 FUNCTION TAm9513a::DigitalOut()
  0    0  290 FUNCTION TAm9513a::DigitalIn()
  0    0  297 FUNCTION TAm9513a::SelectChip()
  0    0  323 FUNCTION TAm9513a::ArmC()
  0    0  331 FUNCTION TAm9513a::LoadC()
  0    0  339 FUNCTION TAm9513a::LoadAndArmC()
  0    0  347 FUNCTION TAm9513a::DisarmAndSaveC()
  0    0  355 FUNCTION TAm9513a::LatchToHoldC()
  0    0  364 FUNCTION TAm9513a::DisarmC()
  0    0  372 FUNCTION TAm9513a::ClearToggleOut()
  0    0  380 FUNCTION TAm9513a::SetToggleOut()
  0    0  389 FUNCTION TAm9513a::ChooseDataPtr()
  0    0  395 FUNCTION TAm9513a::ReadStatus()
  0    0  412 FUNCTION TAm9513a::WriteCmd()
  0    0  430 FUNCTION TAm9513a::WriteData()
  0    0  461 FUNCTION TAm9513a::ReadData()
*** TER   0%  (  0/ 23) of SOURCE FILE DETECUSE\AM9513.CPP
```

MONITORED SOURCE FILE : DETECUSE\BRAUNPSD.CPP

```
#line 1 ".\INCLUDE\detecuse\braunpsd.h"
  0    0   13 FUNCTION TBraun_Psd::SetParameters()
  0    0   19 FUNCTION TBraun_Psd::GetEnergyRange()
#line 20 "detecuse\braunpsd.cpp"
  0    0  134 FUNCTION TBraun_Psd::TBraun_Psd()
  0    0  232 FUNCTION TBraun_Psd::~TBraun_Psd()
  0    0  259 FUNCTION TBraun_Psd::GetBufferSize()
  0    0  267 FUNCTION TBraun_Psd::Initialize()
  0    0  275 FUNCTION TBraun_Psd::SetDataType()
  0    0  289 FUNCTION TBraun_Psd::PsdReadOut()
  0    0  434 FUNCTION TBraun_Psd::PsdInit()
  0    0  482 FUNCTION TBraun_Psd::PsdStart()
  0    0  515 FUNCTION TBraun_Psd::SetEnergyRange()
  0    0  559 FUNCTION TBraun_Psd::PsdStop()
  0    0  586 FUNCTION TBraun_Psd::BuildOperation()
  0    0  619 FUNCTION TBraun_Psd::ResetDelayTime()
  0    0  637 FUNCTION TBraun_Psd::Look_till_BaseAddr1()
  0    0  658 FUNCTION TBraun_Psd::SynchronHexFile()
  0    0  681 FUNCTION TBraun_Psd::konvert()
  0    0  691 FUNCTION TBraun_Psd::LoadHexFile()
*** TER   0%  (  0/ 18) of SOURCE FILE DETECUSE\BRAUNPSD.CPP
```

MONITORED SOURCE FILE : DETECUSE\C_LAYER.CPP

```
 23    0   54 FUNCTION LibMain()
 23    0   63 FUNCTION WEP()
  0    0   70 FUNCTION dlGetVersion()
  0    0   75 FUNCTION dlGetInstance()
 23    0   80 FUNCTION InitializeCountersDLL()
 23    0   86 FUNCTION GetCounterListPtr()
  0    0   92 FUNCTION dlSetDevice()
  0    0  100 FUNCTION dMeasureStart()
  0    0  105 FUNCTION dMeasureStop()
  0    0  110 FUNCTION dSetExposureValues()
  0    0  115 FUNCTION dGetExposureValues()
  5    0  121 FUNCTION dlGetDevice()
 31    0  127 FUNCTION TDList::GetIdByName()
 28    0  142 FUNCTION dlGetIdByName()
  4    0  147 FUNCTION dlParsingDevice()
 24    0  152 FUNCTION TDList::IsDeviceValid()
 23    0  164 FUNCTION dlIsDeviceValid()
 69    0  169 FUNCTION TDList::ParsingAxis()
  5    5  182 FUNCTION TDList::SetParametersDlg()
*** TER  63%  ( 12/ 19) of SOURCE FILE DETECUSE\C_LAYER.CPP
```

MONITORED SOURCE FILE : DETECUSE\COUNTERS.CPP

```
#line 1 ".\INCLUDE\detecuse\braunpsd.h"
  0   0   13 FUNCTION TBraun_Psd::SetParameters()
  0   0   19 FUNCTION TBraun_Psd::GetEnergyRange()
#line 22 "detecuse\counters.cpp"
 23  23   53 FUNCTION TDList::TDList()
 23  23   67 FUNCTION TDList::~TDList()
 23   0  107 FUNCTION TDList::DeleteDevices()
 23   0  112 FUNCTION TDList::InitializeModule()
524   0  206 FUNCTION TDList::DP()
1E3   0  211 FUNCTION TDList::DP()
 48   0  218 FUNCTION TDList::SetDevice()
 88  88  233 FUNCTION TDevice::TDevice()
 88  88  282 FUNCTION TDevice::~TDevice()
 16   0  299 FUNCTION TDevice::SetExposureValues()
  7   7  313 FUNCTION TDevice::GetExposureValues()
 65   0  334 FUNCTION TDevice::Initialize()
171   0  363 FUNCTION EventHandler()
  1   1  395 FUNCTION TDevice::SetEventHost()
  1   1  403 FUNCTION TDevice::KillEvent()
171   0  414 FUNCTION TDevice::GetEventIntensity()
  1   0  430 FUNCTION TDevice::InitializeEvent()
325 325  450 FUNCTION TDevice::UpdateDisplay()
6E3   0  456 FUNCTION TDevice::GetData()
6E3   0  469 FUNCTION TDevice::MeasureStart()
6E3   0  486 FUNCTION TDevice::MeasureStop()
 30   0  497 FUNCTION TDevice::PollDevice()
 96   0  561 FUNCTION TDevice::PushSettings()
 36   0  571 FUNCTION TDevice::PopSettings()
  0   0  593 FUNCTION TGenericDevice::TGenericDevice()
  0   0  602 FUNCTION TGenericDevice::Initialize()
  0   0  619 FUNCTION TGenericDevice::SetParameters()
  0   0  630 FUNCTION TGenericDevice::MeasureStart()
  0   0  645 FUNCTION TGenericDevice::MeasureStop()
  0   0  657 FUNCTION TGenericDevice::PollDevice()
  0   0  712 FUNCTION TGenericDevice::InitializeEvent()
  0   0  717 FUNCTION EventHandler()
  0   0  722 FUNCTION TGenericDevice::SetSound()
  0   0  733 FUNCTION TEncoder::TEncoder()
  0   0  737 FUNCTION TEncoder::~TEncoder()
  0   0  741 FUNCTION TEncoder::Initialize()
  0   0  753 FUNCTION TEncoder::GetData()
  0   0  762 FUNCTION TEncoder::PollDevice()
 23  23  786 FUNCTION TPsd::TPsd()
 23  23  793 FUNCTION TPsd::~TPsd()
 23   0  809 FUNCTION TPsd::Initialize()
  0   0  888 FUNCTION TPsd::SetSpezificParametersDlg()
  2   0  892 FUNCTION TPsd::GetWidth()
 61  61  902 FUNCTION TPsd::SetAddedChannels()
 56   0  918 FUNCTION TPsd::GetMaximumChannel()
 74   0  925 FUNCTION TPsd::GetData()
115   0 1000 FUNCTION TPsd::GetData()
 80   0 1025 FUNCTION TPsd::MeasureStart()
108   0 1050 FUNCTION TPsd::MeasureStop()
  0   0 1060 FUNCTION TPsd::MeasureStopExternal()
 74   0 1067 FUNCTION TPsd::PollDevice()
 13   0 1137 FUNCTION TPsd::SetParameters()
180   0 1142 FUNCTION TPsd::GetAngleRange()
 91   0 1157 FUNCTION TPsd::GetAngleRange()
  0   0 1162 FUNCTION TPsd::SetAngleRange()
 46   0 1167 FUNCTION TPsd::PushSettings()
  7   0 1174 FUNCTION TPsd::PopSettings()
 74   0 1185 FUNCTION TPsd::PsdReadOut()
```

```
  0   0 1224 FUNCTION TStoe_Psd::TStoe_Psd()
  0   0 1231 FUNCTION TStoe_Psd::~TStoe_Psd()
  0   0 1237 FUNCTION TStoe_Psd::Initialize()
  0   0 1246 FUNCTION TStoe_Psd::PollDevice()
  0   0 1312 FUNCTION TStoe_Psd::PsdReadOut()
  0   0 1363 FUNCTION TStoe_Psd::PsdInit()
  0   0 1411 FUNCTION TStoe_Psd::PsdStart()
  0   0 1447 FUNCTION TStoe_Psd::PsdStop()
  0   0 1490 FUNCTION TStoe_Psd::PsdRead()
  0   0 1626 FUNCTION TRadicon::TRadicon()
  0   0 1672 FUNCTION TRadicon::~TRadicon()
  0   0 1687 FUNCTION TRadicon::Initialize()
  0   0 1756 FUNCTION TRadicon::SetSpezificParametersDlg()
  0   0 1764 FUNCTION TRadicon::MeasureStart()
  0   0 1826 FUNCTION TRadicon::MeasureStop()
  0   0 1841 FUNCTION TRadicon::PollDevice()
  0   0 1908 FUNCTION TRadicon::SetParameters()
  0   0 1931 FUNCTION EventHandler()
  0   0 2000 FUNCTION TRadicon::InitializeEvent()
  0   0 2039 FUNCTION TRadicon::SetSound()
  0   0 2044 FUNCTION TRadicon::FailureOccured()
  0   0 2068 FUNCTION TScsParameters::TScsParameters()
  0   0 2084 FUNCTION TScsParameters::Dlg_OnInit()
  0   0 2092 FUNCTION TScsParameters::Dlg_OnCommand()
  0   0 2112 FUNCTION TScsParameters::CanClose()
  0   0 2157 FUNCTION TScsParameters::LeaveDialog()
  5   5 2163 FUNCTION TCommonDevParam::TCommonDevParam()
  5   0 2169 FUNCTION TCommonDevParam::Dlg_OnInit()
153 144 2184 FUNCTION TCommonDevParam::Dlg_OnCommand()
  8   0 2281 FUNCTION TCommonDevParam::CanClose()
  5   0 2322 FUNCTION TCommonDevParam::LeaveDialog()
*** TER  51%  ( 46/ 91) of SOURCE FILE DETECUSE\COUNTERS.CPP

MONITORED SOURCE FILE : DETECUSE\KISL1.C

  0   0   70 FUNCTION setprm()
  0   0  190 FUNCTION begwrk()
  0   0  295 FUNCTION getinf()
  0   0  470 FUNCTION init_b()
  0   0  588 FUNCTION reset()
*** TER   0%  (  0/  5) of SOURCE FILE DETECUSE\KISL1.C

MONITORED SOURCE FILE : DETECUSE\KMPT1.C

  0   0   44 FUNCTION tr_message()
  0   0  271 FUNCTION rc_message()
  0   0  487 FUNCTION out_byte()
  0   0  558 FUNCTION in_byte()
  0   0  626 FUNCTION status_rd()
  0   0  649 FUNCTION status_wr()
  0   0  675 FUNCTION crnt_time()
*** TER   0%  (  0/  7) of SOURCE FILE DETECUSE\KMPT1.C

MONITORED SOURCE FILE : DETECUSE\TESTDEV.CPP

 23  23   16 FUNCTION Testdev::Testdev()
3E3   0   31 FUNCTION Testdev::PollDevice()
3E3   0   53 FUNCTION Testdev::CalculateIntensity()
1E3   0  106 FUNCTION Testdev::InterPolate()
 23   0  116 FUNCTION Testdev::fillDevData()
*** TER 100%  (  5/  5) of SOURCE FILE DETECUSE\TESTDEV.CPP
```

MONITORED SOURCE FILE : DIFRKMTY\M_ARSCAN.CPP

```
  5   5  151 FUNCTION TAreaScanParameters::TAreaScanParameters()
 11  11  284 FUNCTION TAreaScanParameters::SetDevice()
  5   5  293 FUNCTION TAreaScan::TAreaScan()
  5   5  424 FUNCTION TAreaScan::~TAreaScan()
  8   0  462 FUNCTION TAreaScan::CallLocalAction()
  0   0  528 FUNCTION TAreaScan::rButtonDown()
  1   1  594 FUNCTION TAreaScan::lButtonUp()
  1   1  609 FUNCTION TAreaScan::lButtonDown()
  5   4  621 FUNCTION TAreaScan::InitializeDlg()
  5   0 1069 FUNCTION TAreaScan::CanOpen()
  5   0 1081 FUNCTION TAreaScan::CanClose()
  5   0 1090 FUNCTION TAreaScan::New()
  5   5 1127 FUNCTION TAreaScan::Create()
181   0 1161 FUNCTION TAreaScan::SetTitle()
150 150 1183 FUNCTION TAreaScan::UpdateWnd()
213 213 1200 FUNCTION TAreaScan::SetRanges()
6E3   0 1323 FUNCTION TAreaScan::GetThetaOffset()
 10   0 1384 FUNCTION TAreaScan::SetMeasurementArea()
  3   3 1429 FUNCTION TAreaScan::InitializeTask()
128  94 1646 FUNCTION TAreaScan::CounterSetRequest()
  3   3 1940 FUNCTION TAreaScan::SteeringReady()
  4   4 2022 FUNCTION TAreaScan::ShowSensorContinuous()
  1   1 2113 FUNCTION TAreaScan::CalibratePsd()
  0   0 2166 FUNCTION TAreaScan::ExternSychronized()
  2   2 2206 FUNCTION TAreaScan::Interrupt()
  1   0 2236 FUNCTION TAreaScan::ComposeDB()
  9   0 2379 FUNCTION TAreaScan::SaveDismantleCurve()
  1   0 2484 FUNCTION TAreaScan::DismantleDB()
  6   0 2507 FUNCTION TAreaScan::GetAdditionalColumns()
  0   0 2519 FUNCTION TAreaScan::GetShift()
  6   0 2534 FUNCTION TAreaScan::SaveMeasurementInfo()
  5   0 2668 FUNCTION TAreaScan::SaveFile()
 31   0 2846 FUNCTION TAreaScan::UpdateFile()
  6   0 2924 FUNCTION TAreaScan::SaveReport()
  4   0 3028 FUNCTION TAreaScan::LoadReport()
  9   0 3222 FUNCTION TAreaScan::LoadMeasurementInfo()
  4   0 3465 FUNCTION TAreaScan::LoadOldData()
  1   1 3706 FUNCTION TAquisition::TAquisition()
  1   0 3717 FUNCTION TAquisition::Dlg_OnInit()
 19  19 3738 FUNCTION TAquisition::Dlg_OnCommand()
  1   0 3748 FUNCTION TAquisition::CanClose()
  1   0 3841 FUNCTION TAquisition::LeaveDialog()
*** TER  93%  ( 39/ 42) of SOURCE FILE DIFRKMTY\M_ARSCAN.CPP
```

MONITORED SOURCE FILE : DIFRKMTY\M_CCDSCN.CPP

```
  0   0   23 FUNCTION TAreaScanCCDParameters::TAreaScanCCDParameters()
  0   0   71 FUNCTION TAreaScanCCDParameters::SetDevice()
  0   0   82 FUNCTION TAreaScanCCD::TAreaScanCCD()
  0   0  108 FUNCTION TAreaScanCCD::~TAreaScanCCD()
  0   0  140 FUNCTION TAreaScanCCD::New()
  0   0  150 FUNCTION TAreaScanCCD::Create()
  0   0  183 FUNCTION TAreaScanCCD::SetTitle()
  0   0  205 FUNCTION TAreaScanCCD::rButtonDown()
  0   0  216 FUNCTION TAreaScanCCD::InitializeDlg()
  0   0  234 FUNCTION TAreaScanCCD::InitializeTask()
  0   0  257 FUNCTION TSetupAreaScanCCD::TSetupAreaScanCCD()
  0   0  272 FUNCTION TSetupAreaScanCCD::Dlg_OnInit()
  0   0  310 FUNCTION TSetupAreaScanCCD::Dlg_OnCommand()
  0   0  371 FUNCTION TSetupAreaScanCCD::CanClose()
*** TER   0%  (  0/ 14) of SOURCE FILE DIFRKMTY\M_CCDSCN.CPP
```

MONITORED SOURCE FILE : DIFRKMTY\M_SCAN.CPP

```
#line 1 ".\INCLUDE\difrkmty\m_scan.h"
  3   0   99 FUNCTION TSetupStepScan::LeaveDialog()
#line 24 "difrkmty\m_scan.cpp"
  7   7   93 FUNCTION TScanParameters::TScanParameters()
  7   7  166 FUNCTION TScan::TScan()
  7   7  219 FUNCTION TScan::~TScan()
  5   0  267 FUNCTION TScan::New()
  9   0  283 FUNCTION TScan::CanClose()
  7   7  291 FUNCTION TScan::Create()
305   0  326 FUNCTION TScan::SetTitle()
344 344  346 FUNCTION TScan::SetRanges()
  4   4  357 FUNCTION TScan::InitializeDlg()
470   3  389 FUNCTION TScan::InitializeTask()
303 273  623 FUNCTION TScan::CounterSetRequest()
  4   4  764 FUNCTION TScan::SteeringReady()
  4   4  867 FUNCTION TScan::Interrupt()
  2   0  934 FUNCTION TScan::LoadMeasurementInfo()
  8   0 1080 FUNCTION TScan::SaveMeasurementInfo()
  6   0 1187 FUNCTION TScan::SaveFile()
 73   0 1258 FUNCTION TScan::UpdateFile()
  2   0 1292 FUNCTION TScan::LoadOldData()
  0   0 1432 FUNCTION TScan::rButtonDown()
  0   0 1496 FUNCTION TScan::SaveDataBase()
  3   3 1509 FUNCTION TSetupStepScan::TSetupStepScan()
  3   0 1542 FUNCTION TSetupStepScan::Dlg_OnInit()
246 246 1614 FUNCTION TSetupStepScan::Dlg_OnCommand()
  3   0 1807 FUNCTION TSetupStepScan::CanClose()
  1   1 1961 FUNCTION TSetupContinuousScan::TSetupContinuousScan()
  1   0 1987 FUNCTION TSetupContinuousScan::Dlg_OnInit()
 54  54 2025 FUNCTION TSetupContinuousScan::Dlg_OnCommand()
  1   0 2128 FUNCTION TSetupContinuousScan::CanClose()
  1   1 2278 FUNCTION TSetupDynamicStep::TSetupDynamicStep()
  1   0 2317 FUNCTION TSetupDynamicStep::Dlg_OnInit()
163 163 2325 FUNCTION TSetupDynamicStep::Dlg_OnCommand()
  1   0 2372 FUNCTION TSetupDynamicStep::CanClose()
*** TER  94%  ( 31/ 33) of SOURCE FILE DIFRKMTY\M_SCAN.CPP
```

MONITORED SOURCE FILE : DIFRKMTY\M_dlgdif.CPP

```
  0   0   57 FUNCTION TPsdRemoteSync::TPsdRemoteSync()
  0   0   76 FUNCTION TPsdRemoteSync::Dlg_OnInit()
  0   0   93 FUNCTION TPsdRemoteSync::Dlg_OnTimer()
  0   0  107 FUNCTION TPsdRemoteSync::PrintFileName()
  0   0  117 FUNCTION TPsdRemoteSync::GetFileName()
  0   0  123 FUNCTION TPsdRemoteSync::Dlg_OnCommand()
  0   0  171 FUNCTION TPsdRemoteSync::CanClose()
  0   0  176 FUNCTION TPsdRemoteSync::LeaveDialog()
  1   1  189 FUNCTION TCalibratePsd::TCalibratePsd()
  1   0  227 FUNCTION TCalibratePsd::Dlg_OnInit()
305   0  251 FUNCTION TCalibratePsd::Dlg_OnTimer()
 81  81  301 FUNCTION TCalibratePsd::Dlg_OnCommand()
  0   0  537 FUNCTION TCalibratePsd::Dlg_OnHScrollBar()
  2   0  571 FUNCTION TCalibratePsd::GetBarEgde()
305   0  590 FUNCTION TCalibratePsd::GetBarPos()
  1   0  601 FUNCTION TCalibratePsd::CanClose()
  1   0  620 FUNCTION TCalibratePsd::LeaveDialog()
  1   1  638 FUNCTION TChooseScan::TChooseScan()
  1   0  650 FUNCTION TChooseScan::Dlg_OnInit()
 58  58  696 FUNCTION TChooseScan::Dlg_OnCommand()
  0   0  783 FUNCTION TChooseScan::Dlg_OnLButtonUp()
 10  10  798 FUNCTION TChooseScan::Dlg_OnVScrollBar()
  1   0  837 FUNCTION TChooseScan::CanClose()
  3   3  851 FUNCTION TSetupAreaScan::TSetupAreaScan()
  3   0  881 FUNCTION TSetupAreaScan::Dlg_OnInit()
316 316  972 FUNCTION TSetupAreaScan::Dlg_OnCommand()
  3   0 1302 FUNCTION TSetupAreaScan::CanClose()
  3   0 1642 FUNCTION TSetupAreaScan::LeaveDialog()
  1   1 1649 FUNCTION TSetOffset::TSetOffset()
  1   0 1661 FUNCTION TSetOffset::Dlg_OnInit()
 53  53 1704 FUNCTION TSetOffset::Dlg_OnCommand()
  1   0 1799 FUNCTION TSetOffset::CanClose()
  1   1 1815 FUNCTION TComposeDB::TComposeDB()
  1   0 1830 FUNCTION TComposeDB::Dlg_OnInit()
 15  15 1847 FUNCTION TComposeDB::Dlg_OnCommand()
  2   0 1917 FUNCTION TComposeDB::CanClose()
  1   1 1960 FUNCTION TDismantleDB::TDismantleDB()
```

MONITORED SOURCE FILE : INTERNLS\L_LAYER.CPP

```
  0   0   35 FUNCTION TAbout::LeaveDialog()
 23   0   54 FUNCTION LibMain()
 23   0   65 FUNCTION WEP()
  0   0   72 FUNCTION spGetVersion()
  0   0   77 FUNCTION spGetInstance()
  0   0   99 FUNCTION AboutTheMaker()
  0   0  106 FUNCTION TAbout::TAbout()
  0   0  110 FUNCTION TAbout::Dlg_OnInit()
  0   0  122 FUNCTION TAbout::Dlg_OnCommand()
 72  72  146 FUNCTION SetHKL()
327 327  174 FUNCTION GetHKL()
 23  23  185 FUNCTION TFileLocations::TFileLocations()
1E4   0  191 FUNCTION TFileLocations::GetDirectory()
1E3   0  198 FUNCTION TFileLocations::GetName()
1E4   0  205 FUNCTION TFileLocations::GetHWFile()
1E3   0  213 FUNCTION TFileLocations::GetCFile()
 23   0  222 FUNCTION TFileLocations::GetTestDevFile()
117   0  230 FUNCTION TFileLocations::GetMacroFile()
 23  23  242 FUNCTION InitializeFileLocations()
  0   0  246 FUNCTION GetDirectory()
  0   0  250 FUNCTION GetName()
1E4   0  254 FUNCTION GetHWFile()
1E3   0  258 FUNCTION GetCFile()
 23   0  262 FUNCTION GetTestDevFile()
117   0  266 FUNCTION GetMacroFile()
1E4   0  274 FUNCTION GetFrameHandle()
  2   0  279 FUNCTION GetClientHandle()
  6   0  284 FUNCTION GetMainInstance()
 23   0  289 FUNCTION GetDataBasePtr()
 23   0  295 FUNCTION InitializeSPLibrary()
  8   0  302 FUNCTION ParsingXScanType()
3E3   0  314 FUNCTION CreateDefaults()
2E3 2E3  319 FUNCTION SetInfo()
  3   3  328 FUNCTION SetStaticInfo()
  0   0  338 FUNCTION SetStatus()
  7   0  347 FUNCTION FileOpen()
 12   0  398 FUNCTION FileSave()
 10   0  451 FUNCTION SetFPOnData()
  0   0  504 FUNCTION GetBufferLine()
144   0  542 FUNCTION GetFileLine()
173   0  583 FUNCTION UnitEnum()
  0   0  599 FUNCTION UnitStr()
1E7 1E7  612 FUNCTION Delay()
1E4 1E4  623 FUNCTION DelayTime()
  0   0  631 FUNCTION maxi()
  0   0  635 FUNCTION mini()
  0   0  639 FUNCTION maxl()
  0   0  643 FUNCTION minl()
  0   0  647 FUNCTION maxf()
  0   0  651 FUNCTION minf()
  0   0  655 FUNCTION maxd()
  0   0  659 FUNCTION mind()
*** TER  62%  ( 32/ 52) of SOURCE FILE INTERNLS\L_LAYER.CPP
```

MONITORED SOURCE FILE : INTERNLS\M_MAIN.CPP

```
 23   0  132 FUNCTION MessageLoop()
  0   0  158 FUNCTION TMain::GetVersion()
 23   0  173 FUNCTION WinMain()
  0   0  254 FUNCTION ShowProgramStatus()
570   0  274 FUNCTION DoCommandsFrame()
 69  69  590 FUNCTION DoSize()
4E3   0  625 FUNCTION DoCommandsChild()
206 183  864 FUNCTION DoPaint()
2E3 2E3  877 FUNCTION TMain::BlastStatusLine()
2E3 2E3  973 FUNCTION TMain::DrawStatus()
  0   0 1002 FUNCTION MenuSelect()
 23   0 1075 FUNCTION CreateMdiClientWindow()
1E4   0 1107 FUNCTION FrameWndProc()
1E4   0 1268 FUNCTION WndProc()
 23  23 1387 FUNCTION TMain::TMain()
  0   0 1520 FUNCTION TMain::~TMain()
 23  23 1537 FUNCTION TMain::SaveProfile()
 23  23 1567 FUNCTION TMain::LoadProfile()
 23   0 1580 FUNCTION TMain::LoadMenuBar()
  0   0 1793 FUNCTION TMain::TellMessage()
```

```
183 183 1809 FUNCTION TMain::SetWatchIndicator()
 23  23 1883 FUNCTION TMain::SetParameters()
  0   0 1892 FUNCTION TMain::GetDataDlg()
 24  24 1909 FUNCTION NewWindow()
 24  24 1927 FUNCTION TMDIWindow::TMDIWindow()
 24  24 1940 FUNCTION TMDIWindow::~TMDIWindow()
 10   0 1945 FUNCTION TMDIWindow::New()
 24   0 1958 FUNCTION TMDIWindow::RegisterWnd()
 24  24 1969 FUNCTION TMDIWindow::GetWindowClass()
 11   0 1987 FUNCTION TMDIWindow::SaveFile()
  2   0 2054 FUNCTION TMDIWindow::SaveFileAs()
 24   0 2071 FUNCTION TMDIWindow::CanOpen()
 32   0 2082 FUNCTION TMDIWindow::CanClose()
3E3 3E3 2099 FUNCTION TMDIWindow::UpdateWnd()
1E3   0 2124 FUNCTION TMDIWindow::WholeWnd()
 92  92 2131 FUNCTION TMDIWindow::SetFocus()
 44  44 2136 FUNCTION TMDIWindow::UpdateMenu()
  3   0 2162 FUNCTION TMDIWindow::SetFileName()
  6   6 2172 FUNCTION TMDIWindow::Interrupt()
*** TER  85%  ( 33/ 39) of SOURCE FILE INTERNLS\M_MAIN.CPP
```

MONITORED SOURCE FILE : MOTRSTRG\M_LAYER.CPP

```
  0   0   35 FUNCTION mlGetVersion()
  0   0   40 FUNCTION mlGetInstance()
308   0   52 FUNCTION mGetScanSize()
  2   2   58 FUNCTION mStartMoveScan()
  1   1   70 FUNCTION TC_812ISA::StartCheckScan()
  0   0   79 FUNCTION TC_812GPIB::StartCheckScan()
  1   1   86 FUNCTION TC_832::StartCheckScan()
  2   0   95 FUNCTION mGetMoveScan()
  2   0  101 FUNCTION mGetMoveFinishIdx()
300 300  106 FUNCTION mSavePosition()
 23   0  139 FUNCTION mlInitializeMotorsDLL()
1E3   0  151 FUNCTION mlSetAxis()
 85   0  160 FUNCTION mlGetAxis()
1E4   0  168 FUNCTION mlGetIdByName()
121   0  223 FUNCTION mlParsingAxis()
114   0  228 FUNCTION mlIsAxisValid()
  0   0  267 FUNCTION mlIsServerOK()
1E5   0  272 FUNCTION mlGetAxisNumber()
125   0  277 FUNCTION mlGetOffset()
  1   1  282 FUNCTION mlSetAngleDefault()
  2   2  287 FUNCTION mlOptimizingDlg()
1E4   0  293 FUNCTION mlGetDistance()
486   0  307 FUNCTION mlGetValue()
416   0  332 FUNCTION mlMoveToDistance()
1E4   0  349 FUNCTION mlIsMoveFinish()
  3   3  354 FUNCTION mlSetParametersDlg()
  2   2  359 FUNCTION mlPositionControlDlg()
 23  23  364 FUNCTION mlSaveModuleSettings()
  2   2  369 FUNCTION mlInquireReferencePointDlg()
  0   0  377 FUNCTION mSetLine()
  2   2  385 FUNCTION mSetRelativeZero()
 15   0  393 FUNCTION mIsDistanceRelative()
  0   0  398 FUNCTION mIsRangeHit()
  0   0  403 FUNCTION mMoveByDistance()
1E3   0  414 FUNCTION mMoveToDistance()
2E7   0  431 FUNCTION mIsMoveFinish()
404   0  440 FUNCTION mGetDistance()
445   0  458 FUNCTION mGetDistanceProcess()
 13  13  464 FUNCTION mStopDrive()
2E3   0  469 FUNCTION mGetValue()
 35   0  494 FUNCTION mSetValue()
 15   0  511 FUNCTION mGetUnitType()
 11   0  516 FUNCTION mIsCalibrated()
 12  12  521 FUNCTION mActivateDrive()
 47  47  526 FUNCTION mSetCorrectionState()
113   0  531 FUNCTION mGetAxisName()
 37   0  536 FUNCTION mGetAxisUnit()
145   0  541 FUNCTION mGetSF()
832   0  546 FUNCTION mGetDF()
  0   0  551 FUNCTION mExecuteCmd()
 33  33  556 FUNCTION mPushSettings()
 32  32  561 FUNCTION mPopSettings()
 23   0  604 FUNCTION LibMain()
 23   0  612 FUNCTION WEP()
*** TER  85%  ( 46/ 54) of SOURCE FILE MOTRSTRG\M_LAYER.CPP
```

MONITORED SOURCE FILE : MOTRSTRG\MOTORS.CPP

```
 23  23  119 FUNCTION msSetSimulationType()
 23  23  123 FUNCTION msRegister_C812ISA_Get()
 23  23  127 FUNCTION msRegister_C812ISA_Put()
 23  23  131 FUNCTION msRegister_C832_Get()
 23  23  135 FUNCTION msRegister_C832_Put()
9E7   0  140 FUNCTION C812ISA_Get()
4E4 4E4  172 FUNCTION C812ISA_Put()
7E4   0  191 FUNCTION C832_Get()
8E4 8E4  223 FUNCTION C832_Put()
 23  23  241 FUNCTION TMList::TMList()
  0   0  282 FUNCTION TMList::~TMList()
 23   0  288 FUNCTION TMList::InitializeModule()
271   0  371 FUNCTION TMList::ParsingAxis()
 23   0  443 FUNCTION TMList::SaveModuleSettings()
  1   1  459 FUNCTION TMList::SetAngleDefault()
2E7   0  471 FUNCTION TMList::MP()
3E4   0  477 FUNCTION TMList::MP()
  3   3  485 FUNCTION TMList::SetParametersDlg()
  2   2  496 FUNCTION TMList::InquireReferencePointDlg()
1E3   0  506 FUNCTION TMList::SetAxis()
  2   2  526 FUNCTION TCalibrate::TCalibrate()
  2   2  553 FUNCTION TCalibrate::~TCalibrate()
  2   0  562 FUNCTION TCalibrate::Dlg_OnInit()
627   0  591 FUNCTION TCalibrate::Dlg_OnTimer()
 18   0  632 FUNCTION TCalibrate::Dlg_OnCommand()
  2   0  885 FUNCTION TCalibrate::CanClose()
  2   0  894 FUNCTION TCalibrate::LeaveDialog()
  2   2  909 FUNCTION TMList::PositionControlDlg()
  2   2  919 FUNCTION TPosControl::TPosControl()
  2   0  925 FUNCTION TPosControl::Dlg_OnInit()
 35   0  941 FUNCTION TPosControl::Dlg_OnTimer()
136  86  962 FUNCTION TPosControl::Dlg_OnCommand()
  0   0 1036 FUNCTION TPosControl::HScrollBar()
  4   0 1069 FUNCTION TPosControl::GetBarRange()
 38   0 1075 FUNCTION TPosControl::GetBarPos()
  2   0 1081 FUNCTION TPosControl::LeaveDialog()
  3   3 1092 FUNCTION TMotorParam::TMotorParam()
  3   0 1101 FUNCTION TMotorParam::Dlg_OnInit()
168 151 1117 FUNCTION TMotorParam::Dlg_OnCommand()
  3   0 1189 FUNCTION TMotorParam::LeaveDialog()
  4   0 1195 FUNCTION TMotorParam::CanClose()
  1   1 1278 FUNCTION TOptimizeDC_832::TOptimizeDC_832()
  1   0 1285 FUNCTION TOptimizeDC_832::Dlg_OnInit()
 23   0 1298 FUNCTION TOptimizeDC_832::Dlg_OnCommand()
  1   0 1354 FUNCTION TOptimizeDC_832::CanClose()
  1   0 1362 FUNCTION TOptimizeDC_832::LeaveDialog()
  1   1 1381 FUNCTION TC_812::OptimizingDlg()
  1   1 1391 FUNCTION TOptimizeDC_812::TOptimizeDC_812()
  1   0 1399 FUNCTION TOptimizeDC_812::Dlg_OnInit()
 17  17 1411 FUNCTION TOptimizeDC_812::Dlg_OnCommand()
  1   0 1458 FUNCTION TOptimizeDC_812::CanClose()
  1   0 1479 FUNCTION TOptimizeDC_812::LeaveDialog()
150 150 1498 FUNCTION TMotor::TMotor()
150   0 1597 FUNCTION TMotor::Initialize()
  6   0 1773 FUNCTION TMotor::SetAngleRange()
 35   0 1784 FUNCTION TMotor::PushSettings()
 34   0 1794 FUNCTION TMotor::PopSettings()
  0   0 1809 FUNCTION TMotor::funcd()
  0   0 1816 FUNCTION TMotor::rtsave()
1E3   0 1904 FUNCTION TMotor::Translate()
1E4   0 1962 FUNCTION TMotor::Translate()
427  63 1992 FUNCTION TMotor::SetCorrectionState()
 97   0 2037 FUNCTION TMotor::SetHome()
  0   0 2050 FUNCTION TMotor::MoveByPosition()
  0   0 2057 FUNCTION TMotor::MoveToPosition()
1E3   0 2064 FUNCTION TMotor::MoveToAngle()
  0   0 2078 FUNCTION TMotor::MoveByAngle()
 48   0 2091 FUNCTION TMotor::SetAngleWidth()
1E4   0 2129 FUNCTION TMotor::GetAngle()
153   0 2159 FUNCTION TMotor::SaveSettings()
131 131 2201 FUNCTION TDC_Drive::TDC_Drive()
134   0 2213 FUNCTION TDC_Drive::SaveSettings()
131   0 2219 FUNCTION TDC_Drive::Initialize()
  0   0 2253 FUNCTION TDC_Drive::PushSettings()
  0   0 2259 FUNCTION TDC_Drive::PopSettings()
1E3   0 2265 FUNCTION TDC_Drive::MoveToPosition()
  2   0 2296 FUNCTION TDC_Drive::MoveByPosition()
 60   0 2302 FUNCTION TDC_Drive::SetSpeed()
```

```
185   0 2313 FUNCTION TDC_Drive::GetSpeed()
1E4   0 2325 FUNCTION TDC_Drive::GetPosition()
  0   0 2354 FUNCTION TDC_Drive::GetFailure()
 94  94 2369 FUNCTION TC_812::TC_812()
 96   0 2383 FUNCTION TC_812::SaveSettings()
 96   0 2407 FUNCTION TC_812::ActivateFilterParameters()
 94   0 2427 FUNCTION TC_812::Reset()
  0   0 2439 FUNCTION TC_812::SetMoment()
  0   0 2451 FUNCTION TC_812::GetMoment()
  0   0 2460 FUNCTION TC_812::SetLine()
317   0 2476 FUNCTION TC_812::IsLimitHit()
  0   0 2485 FUNCTION TC_812::IsMoveFinish()
319   0 2501 FUNCTION TC_812::IsIndexArrived()
  2   0 2517 FUNCTION TC_812::StartToIndex()
107   0 2534 FUNCTION TC_812::ActivateDrive()
113   0 2548 FUNCTION TC_812::StopDrive()
  0   0 2578 FUNCTION TC_812::SetDynamicGain()
  0   0 2593 FUNCTION TC_812::GetDynamicGain()
  0   0 2599 FUNCTION TC_812::SetTorque()
  0   0 2611 FUNCTION TC_812::GetTorque()
137   0 2618 FUNCTION TC_812::SetVelocity()
176   0 2633 FUNCTION TC_812::GetVelocity()
 96   0 2640 FUNCTION TC_812::SetAcceleration()
  0   0 2658 FUNCTION TC_812::GetAcceleration()
 96   0 2664 FUNCTION TC_812::SetLimit()
 97   0 2676 FUNCTION TC_812::SetHome()
317   0 2689 FUNCTION TC_812::GetStatus()
  0   0 2728 FUNCTION TC_812::_GetPosition()
  0   0 2764 FUNCTION TC_812::_GetFailure()
  1   0 2805 FUNCTION TC_812::MoveRelative()
1E3   0 2816 FUNCTION TC_812::MoveAbsolute()
  0   0 2837 FUNCTION TC_812GPIB::TC_812GPIB()
  0   0 2882 FUNCTION TC_812GPIB::Initialize()
  0   0 2903 FUNCTION TC_812GPIB::CheckBoardOk()
  0   0 2921 FUNCTION TC_812GPIB::ExecuteCmd()
 94  94 3017 FUNCTION TC_812ISA::TC_812ISA()
 94   0 3063 FUNCTION TC_812ISA::CheckBoardOk()
 97   0 3118 FUNCTION TC_812ISA::SetHome()
2E7   0 3135 FUNCTION TC_812ISA::IsMoveFinish()
2E4   0 3171 FUNCTION TC_812ISA::_GetPosition()
2E7   0 3189 FUNCTION TC_812ISA::_GetFailure()
3E3   0 3207 FUNCTION TC_812ISA::ExecuteCmd()
2E4   0 3350 FUNCTION TC_812ISA::PutChar()
1E4   0 3380 FUNCTION TC_812ISA::GetChar()
 37  37 3405 FUNCTION TC_832::TC_832()
 37   0 3439 FUNCTION TC_832::Reset()
  1   1 3452 FUNCTION TC_832::OptimizingDlg()
 39   0 3462 FUNCTION TC_832::ActivateFilterParameters()
 38   0 3481 FUNCTION TC_832::SaveSettings()
  0   0 3505 FUNCTION TC_832::SetHome()
 40   0 3516 FUNCTION TC_832::ActivateDrive()
 43   0 3528 FUNCTION TC_832::StopDrive()
  0   0 3544 FUNCTION TC_832::StartLimitWatch()
  0   0 3555 FUNCTION TC_832::StopLimitWatch()
  0   0 3565 FUNCTION LimitWatch()
  0   0 3607 FUNCTION TC_832::IsLimitHit()
  0   0 3651 FUNCTION TC_832::IsIndexArrived()
  0   0 3697 FUNCTION TC_832::StartToIndex()
 58   0 3722 FUNCTION TC_832::SetVelocity()
 69   0 3731 FUNCTION TC_832::GetVelocity()
 39   0 3738 FUNCTION TC_832::SetAcceleration()
  0   0 3747 FUNCTION TC_832::GetAcceleration()
  0   0 3753 FUNCTION TC_832::SetLimit()
  0   0 3760 FUNCTION TC_832::GetStatus()
2E3   0 3766 FUNCTION TC_832::_GetPosition()
  0   0 3773 FUNCTION TC_832::_GetFailure()
  1   0 3780 FUNCTION TC_832::MoveRelative()
272   0 3791 FUNCTION TC_832::MoveAbsolute()
1E4   0 3805 FUNCTION TC_832::IsMoveFinish()
 37   0 3815 FUNCTION TC_832::CheckBoardOk()
  0   0 3832 FUNCTION TC_832::ExecuteCmd()
1E4   0 3838 FUNCTION TC_832::Drive628()
1E4   0 3867 FUNCTION Drive628c()
1E4   0 3942 FUNCTION GetWord()
2E3   0 3962 FUNCTION GetDWord()
839 839 3986 FUNCTION PutWord()
453 453 4009 FUNCTION PutDWord()
3E4   0 4035 FUNCTION LM628Ready()
*** TER  76%  (119/156) of SOURCE FILE MOTRSTRG\MOTORS.CPP
```

```
MONITORED SOURCE FILE : MOTRSTRG\MSIMSTAT.CPP

#line 1 ".\INCLUDE\motrstrg\msimstat.h"
  4   0   12 FUNCTION TMSimStatus::get_StatusBox()
#line 9 "motrstrg\msimstat.cpp"
  4   4   13 FUNCTION TMSimStatus::TMSimStatus()
  4   4   19 FUNCTION TMSimStatus::~TMSimStatus()
  8   8   30 FUNCTION TMSimStatus::Size()
  4   4   39 FUNCTION TMSimStatus::Create()
 23  23   59 FUNCTION InitializeMotorsSimulation()
  0   0  185 FUNCTION UnInitializeMotorsSimulation()
*** TER  86%  (  6/  7) of SOURCE FILE MOTRSTRG\MSIMSTAT.CPP

MONITORED SOURCE FILE : SWINTRAC\DLG_TPL.CPP

1E4   0   16 FUNCTION DialogProc()
 45   0   44 FUNCTION TModalDlg::ExecuteDialog()
 45  45   51 FUNCTION TModalDlg::TModalDlg()
 45  45   65 FUNCTION TModalDlg::~TModalDlg()
 13   0   73 FUNCTION TModalDlg::Dlg_OnInit()
1E3 1E3   80 FUNCTION TModalDlg::Dlg_OnCommand()
  0   0  104 FUNCTION TModalDlg::CanClose()
595   0  109 FUNCTION ModelessProc()
  3   3  139 FUNCTION TModelessDlg::TModelessDlg()
  3   3  145 FUNCTION TModelessDlg::~TModelessDlg()
  3   0  151 FUNCTION TModelessDlg::Initialize()
  0   0  160 FUNCTION TModelessDlg::Dlg_OnInit()
 81  81  166 FUNCTION TModelessDlg::Dlg_OnCommand()
  0   0  184 FUNCTION TModelessDlg::CanClose()
*** TER  79%  ( 11/ 14) of SOURCE FILE SWINTRAC\DLG_TPL.CPP

MONITORED SOURCE FILE : SWINTRAC\M_DEVICE.CPP

  7   7   22 FUNCTION TCounterWindow::TCounterWindow()
  7   7   50 FUNCTION TCounterWindow::~TCounterWindow()
  7   0   79 FUNCTION TCounterWindow::CanOpen()
  7   7   93 FUNCTION TCounterWindow::Create()
 14  14  122 FUNCTION TCounterWindow::Size()
3E3   0  135 FUNCTION TCounterWindow::SetTitle()
2E3 2E3  143 FUNCTION TCounterWindow::CounterSetRequest()
  6   6  165 FUNCTION TCounterWindow::SetupLogging()
  0   0  185 FUNCTION TCounterWindow::rButtonDown()
 12  12  207 FUNCTION TCounterWindow::SetFocus()
  0   0  212 FUNCTION TCounterWindow::lButtonDown()
328 317  216 FUNCTION TCounterWindow::Paint()
  1   1  326 FUNCTION TCounterShowParam::TCounterShowParam()
  1   0  333 FUNCTION TCounterShowParam::Dlg_OnInit()
  1   0  346 FUNCTION TCounterShowParam::CanClose()
*** TER  87%  ( 13/ 15) of SOURCE FILE SWINTRAC\M_DEVICE.CPP
```

MONITORED SOURCE FILE : SWINTRAC\M_DLG.CPP

```
  0   0   29 FUNCTION TGetData::TGetData()
  0   0   35 FUNCTION TGetData::Dlg_OnInit()
  0   0   46 FUNCTION TGetData::CanClose()
  1   1   57 FUNCTION TPsdParameters::TPsdParameters()
  1   0   62 FUNCTION TPsdParameters::Dlg_OnInit()
 35  35   79 FUNCTION TPsdParameters::Dlg_OnCommand()
  1   0  141 FUNCTION TPsdParameters::CanClose()
  1   0  163 FUNCTION TPsdParameters::LeaveDialog()
  0   0  169 FUNCTION TExecuteCmd::TExecuteCmd()
  0   0  174 FUNCTION TExecuteCmd::Dlg_OnInit()
  0   0  183 FUNCTION TExecuteCmd::Dlg_OnCommand()
  0   0  218 FUNCTION TExecuteCmd::LeaveDialog()
  3   3  225 FUNCTION TMeasurementParam::TMeasurementParam()
  3   0  229 FUNCTION TMeasurementParam::Dlg_OnInit()
  3   0  255 FUNCTION TMeasurementParam::CanClose()
  5   5  282 FUNCTION TAngleControl::TAngleControl()
  5   0  296 FUNCTION TAngleControl::Dlg_OnInit()
612 612  387 FUNCTION TAngleControl::Dlg_OnTimer()
405 405  426 FUNCTION TAngleControl::Dlg_OnCommand()
  3   3  759 FUNCTION TAngleControl::Interrupt()
115  66  766 FUNCTION TAngleControl::Dlg_OnHScrollBar()
 22   0  809 FUNCTION TAngleControl::GetBarEgde()
 49   0  827 FUNCTION TAngleControl::GetBarPos()
  5   0  835 FUNCTION TAngleControl::LeaveDialog()
*** TER  71%  ( 17/ 24) of SOURCE FILE SWINTRAC\M_DLG.CPP
```

MONITORED SOURCE FILE : TOPOGRFY\M_TOPO.CPP

```
 23  23   43 FUNCTION TTopography::TTopography()
 23  23   63 FUNCTION TTopography::Initialize()
  2   2   87 FUNCTION TTopographyExecute::TTopographyExecute()
  2   0  120 FUNCTION TTopographyExecute::Dlg_OnInit()
367   1  179 FUNCTION TTopographyExecute::Dlg_OnTimer()
623 623  230 FUNCTION TTopographyExecute::Dlg_OnCommand()
  2   0  586 FUNCTION TTopographyExecute::LeaveDialog()
  2   2  608 FUNCTION TTopographySetParam::TTopographySetParam()
  2   0  629 FUNCTION TTopographySetParam::Dlg_OnInit()
185 185  673 FUNCTION TTopographySetParam::Dlg_OnCommand()
  2   0  801 FUNCTION TTopographySetParam::CanClose()
  2   0  961 FUNCTION TTopographySetParam::LeaveDialog()
*** TER 100%  ( 12/ 12) of SOURCE FILE TOPOGRFY\M_TOPO.CPP
```

MONITORED SOURCE FILE : WORKFLOW\M_STEERG.CPP

```
#line 1 ".\INCLUDE\workflow\m_steerg.h"
  0    0   30 FUNCTION TCmd::FirstStep()
  0    0   34 FUNCTION TCmd::GetShowData()
  0    0   36 FUNCTION TCmd::ControlStep()
  0    0   38 FUNCTION TCmd::ReadyStep()
  0    0   40 FUNCTION TCmd::Ready()
102    0   42 FUNCTION TCmd::IsPositionValid()
1E4    0   44 FUNCTION TCmd::IsReady()
  4    0   46 FUNCTION TCmd::GetId()
  0    0  408 FUNCTION TEditWindow::IsModified()
  0    0  415 FUNCTION TEditWindow::ClearModify()
  0    0  417 FUNCTION TEditWindow::GetCursor()
  0    0  422 FUNCTION TEditWindow::ClearWindow()
  0    0  424 FUNCTION TEditWindow::DoSearch()
  0    0  426 FUNCTION TEditWindow::ClassName()
  0    0  428 FUNCTION TEditWindow::GetCharacteristic()
#line 24 "workflow\m_steerg.cpp"
 49   49   64 FUNCTION TCmd::TCmd()
 43   43   74 FUNCTION ~TCmd()
  0    0   80 FUNCTION TCmd::DeviceRequest()
  0    0   85 FUNCTION TCmd::GetFailure()
1E4    0   90 FUNCTION TCmd::WakeUp()
413    0  133 FUNCTION TCmd::StartMove()
418    0  153 FUNCTION TCmd::DoAction()
  0    0  194 FUNCTION TInquireCmd::TInquireCmd()
  0    0  200 FUNCTION TInquireCmd::GetShowData()
  4    4  213 FUNCTION TShowValueCmd::TShowValueCmd()
  4    0  235 FUNCTION TShowValueCmd::GetShowData()
  0    0  247 FUNCTION TSaveDataCmd::TSaveDataCmd()
  0    0  268 FUNCTION TSaveDataCmd::GetShowData()
  0    0  292 FUNCTION TSetFileNameCmd::TSetFileNameCmd()
  0    0  311 FUNCTION TSetFileNameCmd::GetShowData()
  0    0  332 FUNCTION TChooseDeviceCmd::TChooseDeviceCmd()
  0    0  354 FUNCTION TChooseDeviceCmd::GetShowData()
 12   12  372 FUNCTION TMoveToPointCmd::TMoveToPointCmd()
 12    0  403 FUNCTION TMoveToPointCmd::GetShowData()
 12    0  414 FUNCTION TMoveToPointCmd::ReadyStep()
  6    6  422 FUNCTION TLoadPointCmd::TLoadPointCmd()
  6    0  439 FUNCTION TLoadPointCmd::GetShowData()
  2    2  452 FUNCTION TCalculateCmd::TCalculateCmd()
  2    0  484 FUNCTION TCalculateCmd::GetShowData()
  2    2  498 FUNCTION TSetWidthCmd::TSetWidthCmd()
  2    0  505 FUNCTION TSetWidthCmd::GetShowData()
  7    7  519 FUNCTION TChooseAxisCmd::TChooseAxisCmd()
  6    0  527 FUNCTION TChooseAxisCmd::GetShowData()
  0    0  544 FUNCTION TSetupScanCmd::TSetupScanCmd()
  0    0  572 FUNCTION TSetupScanCmd::GetShowData()
  5    5  592 FUNCTION TScanCmd::TScanCmd()
105    0  811 FUNCTION TScanCmd::GetShowData()
  5    0  878 FUNCTION TScanCmd::FirstStep()
145    0  890 FUNCTION TScanCmd::ControlStep()
  5    0  960 FUNCTION TScanCmd::ReadyStep()
  1    1  981 FUNCTION TControlFlankCmd::TControlFlankCmd()
  0    0  997 FUNCTION TControlFlankCmd::~TControlFlankCmd()
 63    0 1005 FUNCTION TControlFlankCmd::GetShowData()
  1    0 1021 FUNCTION TControlFlankCmd::FirstStep()
 62    0 1050 FUNCTION TControlFlankCmd::ControlStep()
  0    0 1139 FUNCTION TControlFlankCmd::ReadyStep()
  6    6 1151 FUNCTION TGotoIntensityCmd::TGotoIntensityCmd()
  6    6 1178 FUNCTION TGotoIntensityCmd::~TGotoIntensityCmd()
129    0 1184 FUNCTION TGotoIntensityCmd::GetShowData()
  6    0 1202 FUNCTION TGotoIntensityCmd::FirstStep()
111    0 1216 FUNCTION TGotoIntensityCmd::ControlStep()
  6    0 1271 FUNCTION TGotoIntensityCmd::ReadyStep()
  3    3 1303 FUNCTION TGotoPeakCmd::TGotoPeakCmd()
  3    3 1318 FUNCTION TGotoPeakCmd::~TGotoPeakCmd()
 21    0 1325 FUNCTION TGotoPeakCmd::GetShowData()
  3    0 1342 FUNCTION TGotoPeakCmd::FirstStep()
 12    0 1360 FUNCTION TGotoPeakCmd::ControlStep()
  3    0 1425 FUNCTION TGotoPeakCmd::ReadyStep()
  1    1 1456 FUNCTION TAreaScanCmd::TAreaScanCmd()
  0    0 1517 FUNCTION TAreaScanCmd::GetShowData()
  1    0 1553 FUNCTION TAreaScanCmd::FirstStep()
 45    0 1563 FUNCTION TAreaScanCmd::ControlStep()
  1    0 1611 FUNCTION TAreaScanCmd::ReadyStep()
 23   23 1622 FUNCTION TSteering::TSteering()
 23    0 1637 FUNCTION TSteering::Initialize()
 15    0 1657 FUNCTION TSteering::Reset()
```

```
 37   0 1674 FUNCTION TSteering::StartUp()
 34  34 1683 FUNCTION TSteering::Visualising()
 10   0 1701 FUNCTION TSteering::StartCmdExecution()
 49   0 1714 FUNCTION TSteering::StartCmdExecution()
 11   0 1788 FUNCTION TSteering::ToggleInterrupt()
102   0 1830 FUNCTION TSteering::IsPositionValid()
1E4   0 1838 FUNCTION TSteering::WakeUp()
395   0 1876 FUNCTION TSteering::DeviceRequest()
 47   0 1907 FUNCTION TSteering::ReadyReaction()
  4   0 1920 FUNCTION TSteering::StartMacroExecution()
 38   0 1941 FUNCTION TSteering::ExecuteNextCmd()
  9   9 1984 FUNCTION TSteering::NotifyCmdReady()
  3   3 1996 FUNCTION TSteering::NotifyMacroReady()
415   0 2007 FUNCTION TSteering::GetIntensity()
406   0 2012 FUNCTION TSteering::GetDistance()
1E4 1E4 2018 FUNCTION RecallSteering()
1E4   0 2036 FUNCTION TSteering::StartTimer()
 65   0 2058 FUNCTION TSteering::StopTimer()
  6   0 2069 FUNCTION TSteering::GetMacroById()
 26   0 2081 FUNCTION TSteering::GetMacroByNumber()
  2   0 2091 FUNCTION TSteering::GetMacroByName()
5E3   0 2103 FUNCTION TSteering::GetFileLine()
896   0 2139 FUNCTION TSteering::ParsingCmdParam()
1E3   0 2188 FUNCTION TSteering::ParsingCmd()
117   0 2475 FUNCTION TSteering::ParsingMacroId()
117   0 2549 FUNCTION TSteering::LoadMacro()
  1   0 2663 FUNCTION TSteering::DeleteMacro()
  1   0 2683 FUNCTION TSteering::LoadMacroByUser()
  9   0 2696 FUNCTION TSteering::GetSigma()
  0   0 2701 FUNCTION TSteering::CallMacro()
  0   0 2718 FUNCTION TSteering::IsMacroCalled()
  0   0 2725 FUNCTION TSteering::ReturnMacroCall()
350 231 2735 FUNCTION TSteering::SendReport()
  1   1 2749 FUNCTION TMacroExecute::TMacroExecute()
  1   0 2758 FUNCTION TMacroExecute::Dlg_OnInit()
 65  65 2781 FUNCTION TMacroExecute::Dlg_OnCommand()
  1   0 2938 FUNCTION TMacroExecute::LeaveDialog()
  0   0 2968 FUNCTION TMacroExecute::CanClose()
  0   0 2973 FUNCTION TEditWindow::SetFocus()
  0   0 2978 FUNCTION TEditWindow::TEditWindow()
  0   0 2989 FUNCTION TEditWindow::~TEditWindow()
  0   0 2993 FUNCTION TEditWindow::Create()
  0   0 3000 FUNCTION TEditWindow::Size()
  0   0 3005 FUNCTION TEditWindow::SetTitle()
  0   0 3021 FUNCTION TEditWindow::Paint()
  0   0 3042 FUNCTION TEditWindow::CanClear()
  0   0 3060 FUNCTION TEditWindow::ReadFile()
  0   0 3066 FUNCTION TEditWindow::NewFile()
  0   0 3079 FUNCTION TEditWindow::SaveFile()
  0   0 3090 FUNCTION TEditWindow::rButtonDown()
  0   0 3104 FUNCTION TSetAdjustmentParam::TSetAdjustmentParam()
  0   0 3109 FUNCTION TSetAdjustmentParam::Dlg_OnInit()
  0   0 3116 FUNCTION TSetAdjustmentParam::Dlg_OnCommand()
  0   0 3129 FUNCTION TSetAdjustmentParam::GetParameter()
  0   0 3139 FUNCTION TSetAdjustmentParam::SetParameter()
  0   0 3149 FUNCTION TSetAdjustmentParam::CanClose()
  0   0 3155 FUNCTION TSetAdjustmentParam::LeaveDialog()
 23  23 3163 FUNCTION TAdjustmentParameter::TAdjustmentParameter()
  0   0 3170 FUNCTION TAdjustmentExecute::TAdjustmentExecute()
  0   0 3178 FUNCTION TAdjustmentExecute::Dlg_OnInit()
  0   0 3206 FUNCTION TAdjustmentExecute::Dlg_OnCommand()
  0   0 3317 FUNCTION TAdjustmentExecute::GetNextTask()
  0   0 3355 FUNCTION TAdjustmentExecute::GetParameter()
  0   0 3360 FUNCTION TAdjustmentExecute::LeaveDialog()
  0   0 3386 FUNCTION TAdjustmentExecute::CanClose()
  0   0 3392 FUNCTION TAdjustmentWindow::TAdjustmentWindow()
  0   0 3422 FUNCTION TAdjustmentWindow::~TAdjustmentWindow()
  0   0 3430 FUNCTION TAdjustmentWindow::CanClose()
  0   0 3437 FUNCTION TAdjustmentWindow::Create()
  0   0 3447 FUNCTION TAdjustmentWindow::SetTitle()
  0   0 3456 FUNCTION TAdjustmentWindow::InitializeTask()
  0   0 3461 FUNCTION TAdjustmentWindow::CounterSetRequest()
  0   0 3478 FUNCTION TAdjustmentWindow::SteeringReady()
  0   0 3491 FUNCTION TAdjustmentWindow::rButtonDown()
*** TER  56%  ( 84/150) of SOURCE FILE WORKFLOW\M_STEERG.CPP
```

Anhang F

Glossar

Absolute Null
: Absolute Motorposition, die per definitionem dem Ursprung des verwendeten (Welt-)Koordinatensystems entspricht. Diese Position wird während der Einrichtung des Motors festgelegt. z.B. könnte man festlegen, das die Absolute Null der Tilt-Achse gerade dort ist, wo der Tisch waagerecht ist.
(vgl. *Relative Null*)

ASCII
: *American Standard Code for Information Interchange*
Dies ist ein Code, bei dem die Zahlen 0 bis 255 für Buchstaben, Zahlen, Interpunktionszeichen und andere Zeichen stehen. Der ASCII-Code ist standardisiert und erleichtert die Übertragung von Text zwischen Computern oder zwischen einem Computer und einem Peripheriegerät.

API
: *Application Program Interface*
Dies ist der Befehlssatz, den eine Anwendung für die Anforderung und Ausführung von Diensten unterer Ebene verwendet, die vom Betriebssystem des Computers bereitgestellt werden.

ATOS
: *Automatisiertes Testen oberflächenbasierter Systeme*
Die Testsuite *ATOS* bildet den Mittelpunkt dieser Arbeit. Alle Testaktivitäten werden von diesem Programm aus gestartet.

ATS

Atomic Testscript
ursprünglich *AutoTest Script* für Testskripte für den in der Studienarbeit entwickelten Kommando-Interpreter *AutoTest*; dient als Lowlevel-Sprache zur Ausführung von Kommandos aus einem HTS-Skript

CASE

Computer Aided Software Engineering
CASE ist die computerbasierte Unterstützung beim Softwareentwicklungs-Prozess. Entsprechende Werkzeuge können in allen Bereichen eines Softwareprojektes wie Management, Administration und technische Realisierung eingesetzt werden.

CVS

Concurrent Versions System
verwaltet Quelltextversionen eines Programmes

CVS-Repository

Alle Quellen des XCTL-Systems und zusätzlicher Komponenten wie Testsysteme (Bsp. *ATOS*) unterstehen einer zentralen Quelltextverwaltung (CVS) auf einem dedizierten Server, auf den alle Studenten des Projektes Zugriff haben.

CTE

Classification Tree Editor
Werkzeug zur systematischen Testfallentwicklung

CCD

Charge Couple Device
Ladungsgekoppeltes Gerät; einzeilig für 1-dimensionale oder zweilzeilig für 2-dimensionale Detektoren

DataDiff

Dieses Werkzeug vergleicht eine Textdatei mit einer Referenzdatei. Spezielle Kommandos (Metainformationen) in der Referenzdatei können dabei die Leseposition und die Behandlung eingelesener Zeichenketten (numerischer vs. textueller Vergleich) beeinflussen.

Direktbetrieb

Bewegung des aktuellen Antriebes mit der eingestellten Fahrgeschwindigkeit von der Ist-Position zur vorgegebenen Soll-Position

DLL
Dynamic Link Library
nachladbare Bibliothek zur Bereitstellung von zusätzlichen Funktionen für eine laufende Applikation

Fahrbetrieb
Bewegung des aktuellen Antriebes mit der eingestellten Fahrgeschwindigkeit solange, wie die Cursortasten (<- oder ->) der Tastatur oder die Endeelemente des Scrollbars gedrückt gehalten werden

GUI
Graphical User Interface
grafische Schnittstelle einer Applikation (Windowsfenster etc.)

Halbwertsbreite
Differenz der Abszissen-Werte (x-Achse, in der Praxis meist Winkel oder Entfernungen) links und rechts eines Ordinatenmaximums (y-Achse, üblicherweise Intensität), bei denen der gemessene Wert nur noch die Hälfte des Maximums beträgt. Gelegentlich findet man den Faktor 1/(Wurzel 2) anstelle von 1/2. Dies ist dann der Fall, wenn der Messwert eine Amplitude darstellt, deren Quadrat proportional zur Intensität ist.

HTS
Highlevel Testscript
Skriptsprache zur Umsetzung von Aktions- und Auswertungsschritten; jedes HTS-Kommando wird auf eine Menge von ATS-Kommandos abgebildet

HWB
Halbwertsbreite
→ siehe *Halbwertsbreite*

LOC
Lines Of Code
Anzahl der Quelltextzeilen eines Programms

Peak
Intensitätsmaximum der untersuchten Strahlung

PSD
Position Sensitive Detector
1-dimensionaler Detektor

RC-Datei
Ressourcen zur Beschreibung von Dialogboxen und Menüs eines C++ Programmes. Positionen und Eigenschaften der Steuerelemente werden in Ressourcen-Editoren der Borland- bzw. Microsoft-Entwicklungsumgebungen definiert und als Textdateien gespeichert, um sie beim Kompilieren in das Projekt eingezubinden.

RCParser
Dieses Werkzeug verwaltet für alle relevanten Oberflächenelmente eines Testobjektes die Zuordnung zwischen numerischen ID's und in Testskripten verwendeten Bezeichnern.

Relative Null
Koordinatenursprung der relativen Motorpositionen. Bestimmt durch ein Offset zur Absoluten Null. (vgl. *Absolute Null*)

Rocking-Kurve
Intensitätsverteilung der untersuchten Strahlung in einer Dimension. Die Halbwertsbreite dieser Kurve dient als Qualitätskriterium bei der Justierung der Probe.

Schrittbetrieb
Bewegung des aktuellen Antriebes mit der eingestellten Schrittweite von der Ist-Position aus. Die Bewegung wird aktiviert, wenn eine der Cursortasten der Tastatur (<- oder ->) betätigt wird bzw. auf dem Scrollbar das linke oder rechte Endeelement angeklickt wird. Die Fahrgeschwindigkeit entspricht der maximalen.

Umgebungssimulation
Umfasst die Simulation von Detektor- und Antriebshardware des XCTL-Systems, um einen Betrieb der Steuerungssoftware außerhalb der physikalischen Arbeitsplätze mit realer Messapparatur zu ermöglichen.

UML
Unified Modelling Language
Sammlung verschiedener Diagrammtypen für unterschiedliche Sichten auf ein SW-System. Beispiel Klassendiagramm: Objekte (Klassen) und ihre statischen Beziehungen.

URF

Uniform Resource File

Datenbank über alle Elemente einer grafischen Benutzerschnittstelle eines Testobjektes. In ihr wird die Zuordnung zwischen numerischen ID's und Bezeichnern zur Verwendung in Testskripten festgehalten.

XCTL

X-Control (X von X-ray)

Bezeichnung für das Steuerprogramm zur Halbleiter-Strukturanalyse

Web-Repository

Sammlung aller Dokumente zum Projekt „Software Sanierung“ auf den Webseiten des Lehrstuhls „Softwaretechnik“

Abbildungsverzeichnis

Literaturverzeichnis

Bücher

[1] E. Dustin, J. Rashka, J. Paul: *Software automatisch testen*,
Springer-Verlag Berlin, 2001

[2] E. Denert: *Software-Engineering*,
Springer-Verlag Berlin, 1992

[3] G.J.Myers: *Methodisches Testen von Programmen*,
R. Oldenbourg Verlag München Wien, 1999

Arbeiten, Berichte und Sonstiges

[4] Atif M. Memon: *A Comprehensive Framework for Testing Graphical User Interfaces*,
University of Pittsburgh, Juli 2001

[5] T. Ostrand, A. Anodide, H. Foster, T. Goradia: *A visual test development environment for GUI systems*,
März 1998, New York, in: Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA-98), ACM Press, S. 82-92

[6] M. L. Hammontree, J. J. Hendrickson, B. W. Hensley: *Integrated data capture and analysis tools for research and testing graphical user interface*,
Mai 1992, New York, in: Proceedings of the Conference on Human Factors in Computing Systems, ACM Press, S. 431-432

[7] L. The: *Stress Tests For GUI Programs*,
September 1992

[8] D. S. Weld: *An introduction to least commitment planning*,
AI Magazine 15, 4 (1994), S. 27-61

[9] D. S. Weld: *Recent advances in AI planning*,
AI Magazine 20, 1 (1999), S. 55-64

[10] Stefan Lützkendorf: *Diplomarbeit: Softwaretest in Reverse Engineering-Prozessen*,
Dezember 2001

[11] Stephan Berndt, Jens Ullrich: *Diplomarbeit: Diffraktometrie/ Reflektometrie-Komponente für ein Programm zur Halbleiter-Strukturanalyse*,
Oktober 2001

[12] Johann Letzel, Jens Hanisch: *Studienarbeit: Automatisierter Regressionstest des XCTL-Systems*,
November 2001

[13] ANSI/IEEE Std 829-1983: *IEEE Standard for Software Test Documentation*,
The Institute of Electrical and Electronics Engineers, Inc

[14] Hans-Martin Hörcher: *Testfallspezifikation mittels Message Sequence Charts*,
Treffen der GI-Fachgruppe 2.1.7 (Test, Analyse und Verifikation von Software), Mai 2000, Sankt Augustin, in: GI Softwaretechnik-Trend, Band 20 Heft 2, Mai 2000

[15] Eike Hagen Riedemann: *ISDN-Driver-Test, Eine Testapplikation im Bereich ISDN-Kommunikation (über eine CAPI-Schnittstelle)*,
Treffen der GI-Fachgruppe 2.1.7 (Test, Analyse und Verifikation von Software), Mai 2000, Sankt Augustin, in: GI Softwaretechnik-Trend, Band 20 Heft 2, Mai 2000

[16] J. Wegener, A. Krämer, E. Lehmann: *Automatische Testfallgenerierung mit dem CTL XL*,
Treffen der GI-Fachgruppe 2.1.7 (Test, Analyse und Verifikation von Software), Februar 2001, Elmshorn, in: GI Softwaretechnik-Trend, Band 21 Heft 1, Februar 2001

[17] Eckard Lehmann, DaimlerChrysler AG: *Time Partition Testing: Systematischer Test eingebetteter Systeme*,
Treffen der GI-Fachgruppe 2.1.7 (Test, Analyse und Verifikation von Software), Oktober 2001, Erlangen, in: GI Softwaretechnik-Trend, Band 21 Heft 3, November 2001

Internet

[18] K. Bothe und Studenten: *Entwicklerdokumente nach Phasen*, https://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJEKT98/produktdoku/entwicklerdoku/navigationstabelle.htm

[19] J. Letzel: *ATOS - Benutzerleitfaden*, https://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJEKT98/werkzeuge/BL_ATOS.pdf

[20] J. Letzel: *RCParser - Benutzerleitfaden*, https://www.informatik.hu-berlin.de/Institut/struktur/softwaretechnikII/lehre/PROJEKT98/werkzeuge/BL_RCParser.pdf

[21] Testwell Oy: *Homepage der Firma Testwell Oy*, http://www.testwell.fi/

[22] ApTest: *Homepage der Firma ApTest*, http://www.aptest.com/

[23] Methods Tools: *Homepage mit Sammlung nützlicher Testwerkzeuge*, http://www.methods_tools.com/tools/frames_testing.html

[24] Mercury Interactive: *Produktbeschreibung von WinRunner aus dem Hause Mercury Interactive*, http://www.mercuryinteractive.de/products/winrunner/

[25] Irfan Skiljan: *Homepage zum Freeware-Grafikviewer IrfanView*, http://www.irfanview.com

Stichwortverzeichnis