



Projektarbeit:
“html^AT_EX” - Konvertierungssoftware
–Ausarbeitung–

von:

Kaiser, Björn

Mühlendamm 6

24937 Flensburg

bjoern-kaiser@versanet.de

Matrikel-Nr.: 371658

und

Baß, Björn

Ritterstraße 28

24939 Flensburg

b-bass@versanet.de

Matrikel-Nr.: 341125

Betreuer: Prof. Dr. Hans Werner Lang

SoSe 2011-I

Fachbereich Technik

Fachhochschule Flensburg

Abgabetermin: 23.03.2011

Inhaltsverzeichnis

1	Einleitung - Motivation	1
2	Durchführung der Projektarbeit	1
2.1	Rahmenbedingungen und Tools	1
2.2	Implementierungsphase	3
2.3	Reflektion	5
3	Implementierung der Konvertierung	5
3.1	Funktionsumfang	6
3.1.1	Konsolenanwendung	6
3.1.2	Grafische Oberfläche	6
4	Ausblick	6
A	Erklärung	8

Abbildungsverzeichnis

1	GUI	4
2	GUI-Variante	4
3	Einstellungen	5
4	BILDUNTERSCHRIFT	6

Listings

1	Pfadweiche aus documentreader.cpp	3
---	---	---

1. Einleitung - Motivation

Das Ziel dieses Projektes ist eine möglichst flexible Anwendung zu schaffen, die aus einem Markup in eine andere per XML¹-definierbare Syntax konvertieren kann.

Dies wurde am Beispiel von der Konvertierung von JavaDoc - generiertem HTML-Code und einer anschließenden Umwandlung zu \LaTeX ² verfolgt.

Um möglichst menschenlesbar weitere Konvertierungsszenarien umsetzen zu können wurden folgende Festlegungen getroffen:

- Die Eingabesemantik wird in einer XML-Datei beschrieben und soll die Umsetzung des Ausgangscodes in eine (pseudo)-HTML Semantik beschreiben.
- Die Ausgabesemantik wird ebenso beschrieben und definiert die Konvertierung in das Zielformat.
- Diese beiden Konfigurationsdateitypen müssen im Rahmen ihrer Syntax in der jeweils inline verfassten DTD³ frei beschrieben werden können.

2. Durchführung der Projektarbeit

2.1. Rahmenbedingungen und Tools

Betriebssystem Es wurde parallel unter *Windows 7*⁴ und *Linux*⁵ entwickelt. Dies war erforderlich, da die notwendige Screenreader-Software unter Linux nicht verfügbar ist und keine Alternative unseren Anforderungen genügte.

Sprache Um einerseits hohe Plattformunabhängigkeit und andererseits Performanz zu erreichen wurde das Projekt in der C++ Klassenbibliothek *Qt*⁶ erstellt.

IDE Es wurde die Entwicklungsumgebung⁷ *Qt Creator*⁸ benutzt.

Lokalisierung Es wurden Übersetzungen in den Sprachen Englisch und Deutsch erstellt, die zur Laufzeit gewechselt werden können.

Interface Das Programm ist sowohl über ein grafisches Interface als auch als per Skript bedienbar und verwaltet die Programmeinstellungen in einer XML-Datei.

¹Xtensible Markup Language

²Lamport **TeX** - umfangreiche Sammlung von TeX-Makros

³Dokumenttypdefinition

⁴Microsoft® und Windows® sind eingetragene Marken der Microsoft Corporation.

⁵Linux® ist ein eingetragenes Markenzeichen von Linus Torvalds

⁶Qt® ist ein eingetragenes Markenzeichen der Nokia Corporation. <http://qt.nokia.com/>

⁷engl. Integrated Development Environment

⁸<http://qt.nokia.com/products/developer-tools>

Dokumentation Die Entwicklerdokumentation der Implementierungsdetails wurde aus dem Quellcode mithilfe des freien Dokumentationswerkzeuges *Doxygen*⁹ generiert. Da die Dokumentation des Quelltextes ausführlich ist werden in dieser Ausarbeitung vorwiegend die Arbeitsweisen, Überlegungen und Konzepte besprochen. Für Implementierungsdetails sei auf den Anhang (Seite 7) oder die [Entwicklerdokumentation](#) verwiesen, die auch alle Klassendiagramme enthält und die Kommentare vom Quellcode getrennt übersichtlich darstellt und durchsuchbar ist.

Konventionen Als Dokumentationssprache wurde Englisch gewählt. Die Headerdateien (Klassendeklarationen `.h`) beginnen mit einer allgemeinen Beschreibung der Funktion der Klasse und werden durch die speziellen Kommentare zu den einzelnen Mitgliedern ergänzt. So können aus den Headerdateien alle Informationen gewonnen werden, die für die Benutzung der Klasse relevant sind.

Die Klassendefinitionsdateien (`.cpp`) enthalten an einigen Stellen zusätzliche Kommentare, die die Implementierung dieser Klasse betreffen, um die Weiterentwicklung zu ermöglichen.

Versionierung Für die Versionierung wurde das verteilte Versionskontrollsystem *Git*¹⁰ verwandt. Als Host dient der spezialisierte Webhosting-Dienst *GitHub*¹¹.

Unit Tests Nach einigen Problemen mit der Qt-eigenen Komponente für Unit Tests (QTestLib) haben wir auf Unit Tests vorerst verzichtet. (Nach einmaligem Kompilieren einer beliebigen Klasse, verursachte das Hinzufügen des Makros `Q_OBJECT`, das für die Verwendung der QTestLib erforderlich ist, unter Linux einen Fehler beim Linker - dies ist leider erst am Ende des Projektbearbeitungszeitraumes durch eine Weiche in der `.pro`-Datei von uns gelöst worden, die den Qt spezifischen Präprozessor MOC¹² zu korrektem Verhalten veranlassen konnte.)

Webpräsenz Auf <http://opus4711.github.com/htmlatex/> können der Quelltext, die [Entwicklerdokumentation](#), das [Pflichtenheft](#) und die [Ausarbeitung](#) (dieses Dokument) in einem [zip](#) oder [tar-Archiv](#) heruntergeladen werden.

Eingabehilfen Als Screenreader wurde *Dolphin*¹³ eingesetzt.

Issues Als Issue-Tracking-System wurde die Issues-Komponente von *GitHub* eingesetzt.

⁹<http://www.stack.nl/~dimitri/doxygen/index.html>

¹⁰<http://git-scm.com/>

¹¹<http://github.com/>

¹²meta object compiler)

¹³Dolphin® ist ein eingetragenes Markenzeichen der Dolphin Computer Access Ltd.
<http://www.yourdolphin.com/>

2.2. Implementierungsphase

Das Projekt wurde mithilfe der Extreme Programming-Methode teils in paralleler Einzelarbeit auf Personen- und Aufgaben(Issue)-bezogenen Entwicklungszweigen, teils gemeinsam umgesetzt.

Zu Beginn wurde auf dem Hauptentwicklungszweig "master" der noch funktionslose Prototyp der Anwendung erstellt und die grafische Oberfläche entworfen. Im Folgenden wurden Teilfunktionalitäten in getrennten Branches in kurzen Zyklen (zwischen 30 Min und 5 Std. Programmierzeit) implementiert. Nach gegenseitiger Absprache wurde der Branch dann auf den beiden Betriebssystemen getestet und mit dem Hauptentwicklungszweig verschmolzen ("merge").

Dateisysteme Um Plattformunabhängig zu bleiben, mussten für die Ermittlung der Dateipfade Anpassungen vorgenommen werden.

Listing 1: Pfadweiche aus documentreader.cpp

```
111         else
112             // windows
113             myFileInfo = QFileInfo(_indexFileInfo.absolutePath()
114                                 + QDir::separator() + "\
                                →new_node->getAttributes()["href"]);
115         if (_includeSubDocuments)
116             _documentStack.push(new DocumentData(myFileInfo, new_node, _fileType));
117         if(Settings::DEBUG)
118         {
```

Die Verwaltung der unterschiedlichen Zeilenenden übernahm Git.

Kodierung Die Kodierungsfunktionen von Qt lieferten entgegen der Dokumentation nicht auf beiden Systemen die angestrebten Ergebnisse.

Wurde ein QTextStream mit setCodec("UTF-8") festgelegt, konnte er UTF-8 Text mit und ohne BOM¹⁴ nicht korrekt verarbeiten. Erst die auf einen QString angewandte Methode toLatin1() konnte plattformübergreifend die von uns genutzte UTF-8 Auswahl korrekt darstellen. Uns ist bewusst, dass die Latin1-Codierung in diesem Bereich weitestgehend deckungsgleich ist, dies aber für größtmögliche Flexibilität wohl nur ein Workaround ist. Bei der Verarbeitung der Übersetzungsdateien für dieses Projekt mittels QTextCodec und der Methode codecForName("utf8") trat dieses seltsame Verhalten nicht auf.

Interface Bei der Gestaltung der Kommandozeilenversion haben wir uns in vielen Fällen an den GNU Coding Standards¹⁵ orientiert.

¹⁴Byte Order Mark

¹⁵<http://www.gnu.org/prep/standards/standards.html>

```
bjoern@lotos:~/projekt/htmlatex> ./htmlatex
usage: htmlatex INPUTFILE FORMAT INPUTDEFINITION OUTPUTFILE FORMAT OUTPUTDEFINITION [-g|--gui]
e.g. htmlatex index.html javadoc input_javadoc.xml manual.tex tex output_tex.xml -g
    -g, --gui          Launch the GUI
    -h, --help         Show some examples
    -v, --verbose      Verbose mode
    -lang=de, -lang=en Sets the language to German or English

See the "README" file for further information.
```

Abbildung 1: GUI

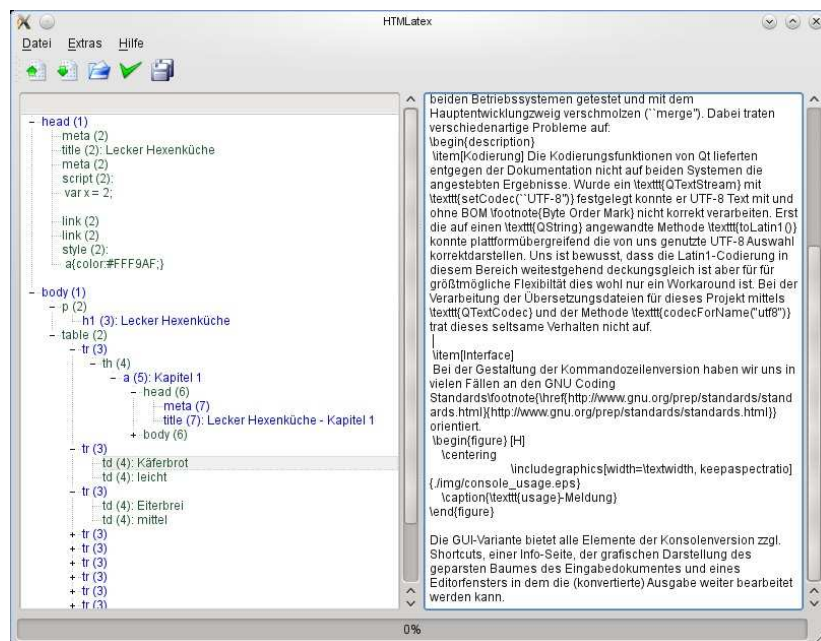


Abbildung 2: GUI-Variante

Die GUI-Variante bietet alle Elemente der Konsolenversion zzgl. Shortcuts, einer Info-Seite, einem Fortschrittsbalken, der grafischen Darstellung des geparsten Baumes des Eingabedokumentes und eines Editorfensters in dem die (konvertierte) Ausgabe weiter bearbeitet werden kann.

Dank des Dokument-Generators Doxygen ist eine Einbindung der Entwicklerdokumentation dieses Projektes als "Hilfe"-Komponente kein großer Aufwand mehr, da direkt Qt Compressed Help (.qch) ausgegeben werden kann.

Einstellungen Die Einstellungen werden in der Datei `htmlatex_settings.dat` binär serialisiert.

Lokalisierung Zu übersetzende Strings wurden mittels der Qt-Funktion `tr("str")` markiert, mit dem Programm `lupdate` in ein XML-Zwischenformat (.ts) extrahiert und mithilfe des `QtLinguist` für deutsch und englisch übersetzt. Die

Übersetzungen werden dann mittels des Programms `lrelease` in das binäre `.qm`-Format (Qt Message File Format) überführt, das auf hohe lookup-Geschwindigkeit optimiert ist und von der Programmdatei direkt genutzt wird. Fehlt die gewünschte Sprachdatei, werden die im Quelltext direkt geschriebenen Strings ausgegeben.



Abbildung 3: Einstellungen

externer Programmaufruf Der Pfad zu \LaTeX Programmdatei kann ebenfalls in den Einstellungen eingegeben werden. Für Linux könnte auch ein Aufruf des Kommandos `which` Voreinstellungen liefern.

2.3. Reflektion

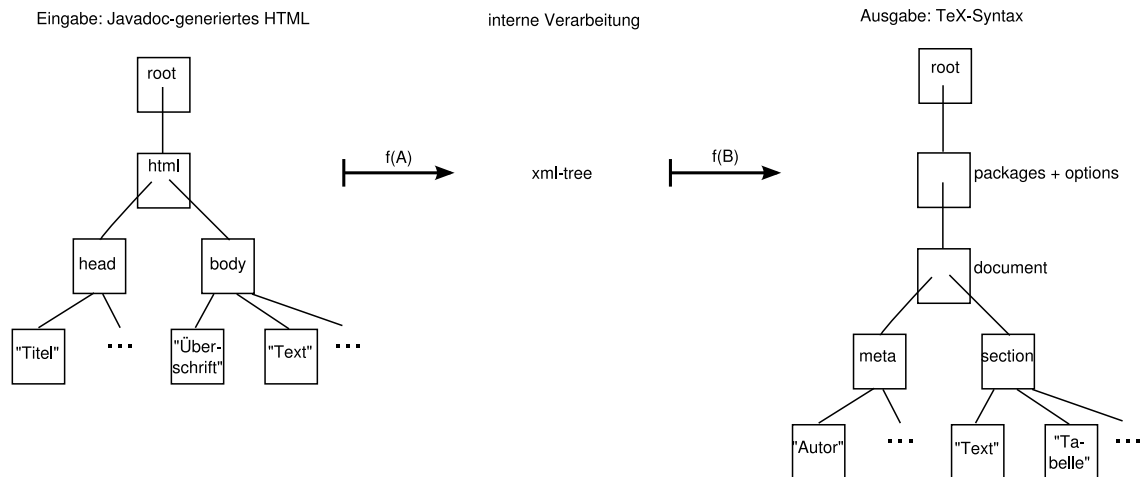
3. Implementierung der Konvertierung

Es wurde die im Pflichtenheft visualisierte Basis für die Konvertierung angestrebt. Hierbei haben wir uns für einen XML-DOM¹⁶-Baum für die interne Verarbeitung festgelegt.

1. Das Eingabeformat steht – im Falle von JavaDoc, wie es z.B. für Java unter <http://download.oracle.com/javase/6/docs/api/overview-summary.html> zu finden ist – leider nicht in einem XML konformen Format bereit. Um es hierin zu überführen, nimmt ein dafür geschriebener Präprozessor die nötigen Umformungen vor. Er wird in einer Hook-Methode vor dem Einlesevorgang ausgeführt und ist leicht zu ersetzen. Er wird aktiv, wenn als Eingabeformat *JavaDoc* ausgewählt wird.

¹⁶Document Object Model

2. Nun beginnt der Einlesevorgang () mithilfe der Eingabe-Definitionsdatei im XML-Format.



3.1. Funktionsumfang

3.1.1. Konsolenanwendung

3.1.2. Grafische Oberfläche

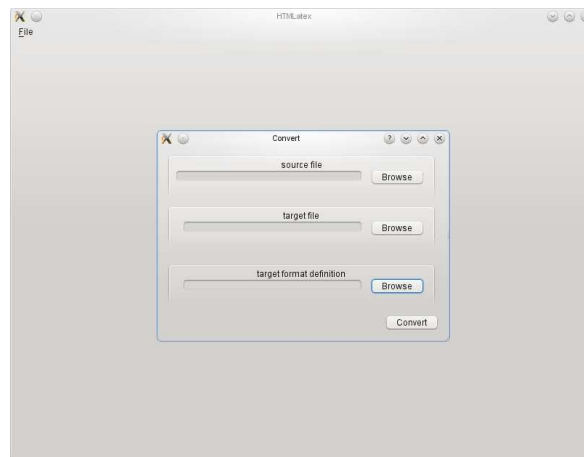


Abbildung 4: BILDUNTERSCHRIFT

4. Ausblick

Neben Verbesserungen am Interface ist es nun interessant zu überprüfen, ob das Programm in der Lage ist in, der Praxis mit mehr Formaten erfolgreich zu arbeiten

und Aussenstehende zu gewinnen, die Formatdefinitionen in XML-Form anpassen.

Zur Senkung des Aufwandes wäre es wohl sehr wünschenswert, wenn dieses z.B. mittels einer Eingabemaske zum Bearbeiten und Erstellen dieser Beschreibungsdateien geschähe. Darüber hinaus würde ein Syntaxhighlighting Tippfehler weiter minimieren.

Der die Anpassungen, die der Präprozessor vorzunehmen hat, damit das Eingabeformat XML-Konform wird könnten auch anpassbar in XML definiert werden.

Der Editor für die Ausgabedateien sollte neben "Suchen&Ersetzen" und einer Schriftgrößen-Einstellung die üblichen Textbearbeitungswerkzeuge zum *Ausschneiden*, *Kopieren* und *Einfügen*, so wie für *Rückgängig* und *Wiederholen* bekommen. `undo()`, `redo()`, `cut()`, `copy()`, `paste()` und `setFontPointSize()` sind im *QTextEdit*-Widget schon enthalten und deshalb auch leicht nachzurüsten.

Die Baumansicht des Eingabedokumentes sollte per Kontextmenü einen speziellen Knoten oder Unterbaum vor dem Übersetzungsvorgang entfernen können, um in Dokumentenstrukturen mit vielen Unterdokumenten die Auswahl begrenzen zu können.

Ebenso ist es erstrebenswert, die maximal zu verfolgende Linktiefe begrenzen zu können und Linkschleifen zu vermeiden.

A. Erklärung

Hiermit erklären wir, dass das Projekt $\text{html}\text{\LaTeX}$ von uns selbständig und ohne Hilfe Dritter erarbeitet und realisiert wurde.

Björn Kaiser

Björn Baß