# SOFTWARE REQUIREMENTS SPECIFICATION

## für Codegenerator GetOptGen

## Programmentwurf C/C++ 2022 TIK/TIM/TIS/TIT21

**Draft 0.9**

**Version 0.9 not yet approved**

**Erstellt durch Th. Staudacher und A. Maus**

# Inhaltsverzeichnis

# Revision History

- Version 0.9: Diese Version ist noch nicht final. Sie ist noch nicht mit den Kursen abgeglichen und einzelne Requirements können sich noch ändern.

# 1 Einführung

## 1.1 Gültigkeit

Diese Version ist Version 0.9 . Sie ist noch nicht final. Sie ist noch nicht mit den Kursen abgeglichen und einzelne Requirements können sich noch ändern.

## 1.2 Gruppengröße

Die empfohlene Gruppengröße für die Bearbeitung dieses Projekts sind 4 Teilnehmer. Die maximale Gruppengröße liegt bei 5 Teilnehmern. Der Prüfungsumfang ändert sich nicht durch kleinere Gruppengrößen.

## 1.3 Aufgabenstellung

Erstellen Sie einen Codegenerator der eine Klasse zum Auswerten von Übergabeparametern erzeugt. Die Optionen sollen hierbei durch eine XML-Datei an das Programm übergeben werden. Desweiteren sind in der XML-Datei auch Angaben zum Autor, Beispiele, Beschreibungen und Ablageorte hinterlegt. Je nach den Vorgaben aus der XML-Datei soll die mit dem Codegenerator erzeugte Klasse entweder direkt instanziierbar oder abstrakt sein. Die Formatierung der erzeugten Header- und Source Datei ist nicht Umfang des Programmentwurfs, diese kann unter Zuhilfenahme des Kommandozeilenprogramms astyle erfolgen. Der Aufruf von astyle findet ebenfalls nicht im Programmentwurf statt.

# 2 Allgemeine Beschreibung

## 2.1 Funktionale Anforderungen

Funktionale Anforderungen beschreiben was das Programm ausführen soll. Im Gegensatz zu nichtfunktionalen Anforderungen sind sie direkt in der Software zu finden, d.h. es wird kein Tooling beschrieben sondern was im Programmablauf zu beachten ist.

### 2.1.1 Genereller Ablauf

Dem Codegenerator soll per Kommandozeilenparameter eine XML-Konfigurationsdatei übergeben werden. Der Codegenerator soll diese XML-Konfigurationsdatei einlesen und basierend auf den darin enthaltenen Einstellungen Code-Dateien generieren. Die auf diese Weise generierten Dateien sollen das Grundgerüst für eine neue Applikation bilden.

### 2.1.2 XML-Format

Das Format der XML-Einstellungsdatei ist wie folgt vorgegeben (siehe Example.xml). Zur Reduktion der Komplexität wird es sich anbieten, Sonderzeichen im XML-Text bereits fertig für C/C++ zu „escapen“.

#### GetOptSetup

Das Attribut SignPerLine stellt die Anzahl der Buchstaben pro Zeile ein. Typisch für die Kommandozeile ist der Defaultwert 79.

#### Author

Die Attribute Name, Phone und Mail beinhalten Informationen über den Autor, die beim Hilfetext mit ausgegeben werden sollen.

#### HeaderFileName

HeaderFileName hat keine Attribute. Zwischen Start- und End-Tag steht der geplante Dateiname der anzulegenden Header-Datei.

#### SourceFileName

SourceFileName hat keine Attribute. Zwischen Start- und End-Tag steht der geplante Dateiname der anzulegenden Quellcode-Datei.

### NameSpace

NameSpace hat keine Attribute. Zwischen Start- und End-Tag steht der geplante Name des anzulegenden Namespace in der sich die generierte Klasse befindet.

### ClassName

ClassName hat keine Attribute. Zwischen Start- und End-Tag steht der geplante Name der anzulegenden Klasse.

### OverAllDescription

OverAllDescription hat keine Attribute. Zwischen Start- und End-Tag stehen 1 oder mehrere Block-Tags. Block hat keine Attribute. Zwischen Start- und End-Tag steht ein Textblock, der das Programm beschreibt.

### SampleUsage

SampleUsage hat keine Attribute. Zwischen Start- und End-Tag stehen 1 oder mehrere Sample-Tags. Sample hat keine Attribute. Zwischen Start- und End-Tag steht ein möglicher Programmaufruf.

### Options

Options hat keine Attribute. Zwischen Start- und End-Tag stehen 1 oder mehrere Option-Tags. Option hat die Attribute Ref (Optional), ShortOpt, LongOpt (minimum 1 von beiden muss belegt sein), Exclusion, ConnectToInternalMethod, HasArguments und Description.

- Ref ist ein Integerwert und dient dazu die einzelnen Attribute referenzieren zu können, zum Beispiel wenn sie sich gegenseitig ausschließen sollen.
- ShortOpt definiert die Parameter die nur aus einem einzelnen Buchstaben bestehen können.
- LongOpt definiert die Parameter die aus ganzen Worten bestehen können.
- Exclusion listet die Referenzen der Optionen mit der diese Option nicht gemeinsam aufgerufen werden darf
- ConnectToInternalMethod definiert die Methode die beim Auswerten dieses Optionsparameters aufgerufen wird.
- HasArguments kann die Werte optional und required annehmen. Wenn keine die Option keine Argumente nutzt kann HasArguments weggelasssen werden.
- Description ist der eigentliche Hilfetext der zu der Option angezeigt werden soll.

### 2.1.3 Optionsparameter

#### Darstellung im Hilfe Menü

Um kein aufwendiges sortieren von Hand in der XML Konfiguration durchführen zu müssen soll der Parser die Optionen nach Alphabet sortieren. Prior sind die Buchstaben der Kurzoptionen, dann deren der Langoptionen (ohne führende -/– zu beachten).

{ReqFunc1} Nach Parsen aller Optionen sollen die Optionen nach Alphabet aufsteigend sortiert werden [A-Z]

Es soll eine Hilfetextausgabe realisiert werden. Diese soll als Methode in der Klasse hinterlegt werden. Der Hilfetext soll aus den Optionen generiert werden. Dabei ist die Kurz-, die Langoption, die Beschreibung sowie die Angabe zum Autor und Benutzung unterzubringen. Die Textbreite soll durch die XML-Einstellungen vorgegeben werden.

{ReqFunc2} Es ist eine virtuelle protected Methode printHelp zu erstellen die den Hilfetext auf der Konsole ausgibt.

{ReqFunc3} Der Hilfetext soll die Inhalte Blockformatieren.

{ReqFunc4} Der Hilfetext ist so ein zu fassen, dass er die Angabe aus dem XML Tag SignPerLine nicht überschreitet (Defaultwert 79 Zeichen/Linie).

{ReqFunc5} Der Hilfetext soll am Ende eine Zeile mit den Angaben aus dem XML Tag Author enthalten.

{ReqFunc6} Der Hilfetext soll die Beschreibungen aus dem XML Tag OverAllDescription enthalten.

{ReqFunc7} Der Hilfetext soll die Beispielbenutzung aus dem XML Tag SampleUsage enthalten.

### 2.1.4 Codegenerierung

#### Dateinamen

Die Dateinamen der Header- und der Source Datei sollen aus der XML-Datei entnommen werden. Es gibt dafür keine Vorgaben. Sind diese Tags leer so kann die jeweilige Datei nicht generiert werden. Auf eine komplette Pfadextraktion wird verzichtet da unter Windows die Pfadauflösung anders geregelt ist wie unter Unix. Es handelt sich hierbei nur um reine Dateinamen.

{ReqFunc8} Ist in der XML-Datei der XML Tag HeaderFileName befüllt, soll eine Header-Datei mit diesem Namen erstellt werden.

{ReqFunc9} Ist in der XML-Datei der XML Tag SourceFileName befüllt, soll eine Source-Datei mit diesem Namen erstellt werden.

#### Header Datei

Um die Header Datei mehrfach includieren zu können ist eine Mehrfacheinbindung vorzusehen. Diese soll in die erzeugte Datei eingeschrieben werden dass sie sowohl unter Windows MinGW wie auch unter UNIX GNU-C Compiler funktioniert. Das Kommando #pragma once ist nicht erwünscht.

{ReqFunc10} Die erzeugte Header Datei soll Mehrfacheinbindung unterstützen.

{ReqFunc11} Die Mehrfacheinbindung der erzeugten Header Datei soll kompatibel zum Compiler Windows MinGW wie auch unter UNIX zu GNU-C sein.

### 2.1.5 Allgemeine Dateigenerierungs Regeln

#### Namespaces

Ist in der XML-Datei ein Namespace benannt soll dieser mit in die Erzeugung der Zieldateien geschrieben werden.

{ReqFunc12} Ist in der XML-Datei der XML Tag NameSpace befüllt, soll in der Header-Datei dieser Namespace eingefügt werden.

{ReqFunc13} Ist in der XML-Datei der XML Tag NameSpace befüllt, soll in der Source-Datei dieser Namespace eingefügt werden.

### 2.1.6 Demonstration durch ein Beispielprojekt

#### Inhalte

Es soll eine Beispielapplikation erstellt werden zur der eine XML-Konfiguration erstellt wird. In der XML-Konfiguration soll die gesammte Funktionalität vorgeführt werden. Was das Programm macht ist zweitrangig, die Testimplementierung der erzeugten Klasse steht im Vordergrund.

{ReqFunc14} In der Musterimplementierung soll eine Autorenangabe der Gruppenteilnehmer enthalten sein.

{ReqFunc15} In der Autorenangabe der Gruppenteilnehmer muss keine Telefonnummer enthalten sein.

{ReqFunc16} In der Autorenangabe der Gruppenteilnehmer muss mindestens eine Kontakt-Mail Adresse enthalten sein um Rückfrage seitens Dozent stellen zu können.

{ReqFunc17} In der Musterimplementierung soll ein Namespace enthalten sein.

{ReqFunc18} In der Musterimplementierung soll im Hilfetext die Beschreibung des Programms enthalten sein.

{ReqFunc19} In der Musterimplementierung soll im Hilfetext die Musterbenutzung enthalten sein.

{ReqFunc20} Die erstellte Klasse soll eine beliebige Zahl von Argumenten parsen können die nur von getopt(...) und getopt_long(...) limitiert sind.

{ReqFunc21} Die erzeugte Parser Klasse soll Character vom Type ASCII verarbeiten können.

{ReqFunc22} Die generierte Klasse soll Ihren Speicher selbständig verwalten. Nach Entfernen der Instanz soll der ganze Speicher wieder freigegeben werden.

{ReqFunc23} Das Generierungsprogramm soll den Speicher selbständig verwalten und auch wieder freigeben.

{ReqFunc24} In der Musterimplementierung soll die Option Hilfe mit dem ShortOpt -h und LongOpt –help enthalten und zur internen Methode printHelp (ConnectToInternalMethod) verbunden werden.

{ReqFunc25} In der Musterimplementierung soll die Option Version mit dem ShortOpt -v und LongOpt –version enthalten (Inhalte der Version bitte selbst bestimmen) und zu einer externen Routine verbunden werden (Interface).

{ReqFunc26} In der Musterimplementierung soll mindestens ein Ausschluss (Ref/Exclusion) enthalten sein.

{ReqFunc27} In der Musterimplementierung soll mindestens ein required Parameter mit Typwandlung bool enthalten sein.

{ReqFunc28} In der Musterimplementierung soll mindestens ein optional Parameter mit Typwandlung Integer enthalten sein, wobei der boolwert aus DefaultValue ausgelesen wird.

#### Erstellungsprozess der Musterimplementierung

Es soll eine CMake Datei erstellt werden die die Musterimplementierung sowohl mit MinGw (Windows) als auch mit GNU-C/C++ (UNIX) erstellen kann.

{ReqFunc29} Es soll eine CMake Datei erstellt werden die die Musterimplementierung sowohl under MinGw (Windows) und GNU-C/C++ (UNIX) erstellen kann.

#### Dokumentation der Musterimplementierung

Die Musterimplementierung soll dokumentiert werden. Zusammen mit der erzeugten Header- und Source-Datei können die Dozenten die Funktion der einzelen Teilfunktionen aus der Dokumentation verstehen.

{ReqFunc30} Die Dokumentation im Beispielprojekt soll in doxygenFormat sein.

## 2.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen sind umzusetzen. Im Gegensatz zu funktionalen Anforderungen, sind nichtfunktionale Anforderungen nicht direkt an Features des Programms gebunden sondern beschreiben die Umgebung und das Tooling.

### 2.2.1 Tools und Build Umgebungen

#### Unterstütze Plattformen und Compiler

Um die Software auf den Systemen Windows 7/10/11 und Linux Ubuntu lauffähig zu bekommen ist zwingend erforderlich einen Compiler zu verwenden, der die gleichen Kommandozeilenparameter versteht und frei von Lizenzen ist. Das ist aktuell beim MSVC2017 Compiler nicht gegeben.

{ReqNonFunc1} Die Software soll auf MinGW64 für Windows complilerbar sein

{ReqNonFunc2} Die Software soll mit GNU-C/C++ für UNIX complilerbar sein

#### Erstellung der ausführbaren Datei

Um die Software frei von graphischen Entwicklungsumgebungen erstellen zu können wird ein Makefile benötigt. Dieses kann dann unter mingw32-make (Windows) oder make (UNIX) erstellt werden. Da der Umfang dieses von Hand zu erstellen den Umfang des Programmentwurfs überschreiten würde, wird hier auf CMake als Erstellungstool für das Makefile verwiesen.

{ReqNonFunc3} Das Erstellen der Software soll mit CMake erfolgen

#### Einlesen von XML-Dateien

Eine XML-Datei kann mit einer schier unüberschaubaren Anzahl von Libraries in C/C++ eingelesen werden. Die bekannteste hier ist XERCES von Apache. Apache ist ein kostenloses Framework zum hosten von XHTML Seiten. Die Library beinhaltet die Varianten SAX/SAX2 und DOM. SAX durchläuft das Dokument linear und DOM baut eine Baumstruktur auf. Beides ist geeignet zum auslesen einer XML-Datei. Mit DOM kann auch eine XML-Datei erstellt werden.
Hineweis: Ob die XML-Datei als SAX oder DOM ausgelesen wird bleibt der Gruppe die die Implementierung erstellt überlassen.

{ReqNonFunc4} Zum einlesen der XML Datei muss die Softwarelibrary XERCES-C verwendet werden

#### Dokumentation des Programmentwurf

Um den Dozenten Th. Staudacher und A. Maus die Chance zu geben, die Struktur und Funktionalität des Programmentwurf zu Überblicken ist es dringend gegeben, dass

die Dokumentation des Codes im doxygen-Format erfolgt. Durch gute Dokumentation werden Rückfragen im Fehlerfall minimiert und es können ggf. auch besser Teilpunkte vergeben werden.

{ReqNonFunc5} Die Dokumentation des Codes muss mit dem Syntax von doxygen erfolgen

#### Einlesen der Optionen

Um eine einheitliche Basis zu schaffen und das Programm portabel zu halten wird zum Parsen der Optionen die getopt(_long) Implementierung verwendet. Das implementieren anderer Libraries oder eigengeschriebene Anwendungen ist nicht gestattet.

{ReqNonFunc6} Das Parsen der Optionen soll über die LibC Funktionalität getopt(...) und getopt_long(...) aus getopt.h erfolgen

#### Dateien und Verzeichnisstruktur

Um eine saubere Trennung zwischen Implementierung und Deklarierung zu bekommern ist eine Auftrennung in zwei Verzeichnisse nötig. Unterverzeichnisse in dieser Auftrennung sind ebefalls möglich.

{ReqNonFunc7} Header- und Source Dateien sollen sich in verschiedenen Unterverzeichnissen befinden

{ReqNonFunc8} Header Dateien sollen relativ und nicht absolut includiert werden

Um die Bewertung der Gruppe durch zu führen ist die angefügte Datei students.xml mit den Teilnehmer der Gruppe aus zu füllen.

{ReqNonFunc9} Die beigestellte Datei students.xml soll von der Gruppe vollständig befüllt sein und in der Abgabe enthalten sein (Format utf-8)

#### Programmierung

In dieser Sektion sind die Codierrichtlinien enthalten die dafür sorgen dass der Code der geforderten Qualität entspricht. Diese Codierrichtlinien sind unbedingt einzuhalten. Muss aus irgendeinem Grund davon abgewichen werden so ist dies in einem Kommentar zu begründen.

{ReqNonFunc10} Daten und Methoden die nur einer internen Berechnung dienen (nicht zum Export) sollen gegen externen Aufruf oder auslesen geschützt werden

{ReqNonFunc11} Zeiger sollen immer initialisiert werden, Ausnahmen nur gestattet bei entsprechender Kommentierung

{ReqNonFunc12} Der Gültigkeitsbereich einer Variablen soll beachtet werden

{ReqNonFunc13} Werden logische Verknüpfungen gemacht so sind logische Operatoren zu verwenden

{ReqNonFunc14} Berechnungen in logischen Verknüpfungen sind nicht gestattet

{ReqNonFunc15} Werte die im weiteren Kontext nicht mehr verändert werden sollen, sind konstant zu halten

{ReqNonFunc16} Zeiger die im weiteren Kontext nicht mehr verändert werden sollen, sind konstant zu halten

{ReqNonFunc17} Das wegcasten von const ist nicht gestattet

{ReqNonFunc18} Die Gültigkeit von Variablen soll an Übergabestellen überprüft werden

{ReqNonFunc19} Jede Header- und Source Datei soll am Anfang ein Hinweis auf den Bearbeiter enthalten

{ReqNonFunc20} Das kommentieren des Codes soll in deutscher Sprache erfolgen, optional in Englisch

{ReqNonFunc21} Werden Variablen verwendet die eine fixe Bitlänge benötigen so soll die Definition aus cstdint verwendet werden

{ReqNonFunc22} Berechnungen mit Fließkommazahlen sind nicht gestattet

{ReqNonFunc23} Das verwenden von goto ist nicht gestattet

{ReqNonFunc24} Das verwenden von #pragma ist nur gestattet solange dadurch keine Compilerabhängigkeit entsteht

## 2.3 Optionale Anforderungen

Optinale Requirements fliessen nicht in die Bewertung ein. Es hat sich aber in den letzten Jahren gezeigt, dass eine saubere Entwicklungsumgebung viele Fehler vermeidet. Es ist angeraten einige der optionalen Requirements mit um zu setzen.

### 2.3.1 Tools und Build Umgebungen

#### Libraries für C++

Es kann der ganze Umfang der mitgelieferten Headerdateien (soweit unter Windows und UNIX gleich einsetzbar) verwendet werden. Alternativ muss per Präprozessor jeweils eine Umschaltung erfolgen. Vordefinerte Macros sind hierbei __WIN32__, __UNIX__ etc. Dies sind dann ggf. in der Dokumentation das Compilers nachzuschlagen. Um die volle Funktionalität einer C++ Implementierung auspielen zu können, kann sowohl die STL wie auch Boost als Library verwendet werden.
Anmerkung: Die STL ist Bestandteil des g++ Compilers, Boost hingegen nicht. Diese kann leicht unter UNIX installiert werden (boost-dev). Bei MinGw64 wird geraten gleich ein fertiges Paket aus MinGW64+Boost zu installieren.

{ReqOptFunc1} Die Library STL kann vollumfänglich verwendet werden

{ReqOptFunc2} Die Library Boost kann vollumfänglich verwendet werden

#### Logging von Ereignissen

Um den Bildschirm nicht mit Meldungen zu überfrachten hat es sich eingebürgert, ein Logging zu verwenden. Das ursprünglich für Java entwickelte Log4J wurde in fast alle Programmiersprachen überführt. Im Unterricht wurden zumindestens eines der beiden Logging-Libraries Easylogging++ oder Boost Logging besprochen. Es ist dringend angeraten dieses auch im Programmentwurf ein zu setzen. Es kann auch helfen im Fehlerfall im Dialog mit den Studenten eine lauffähige Version nach Abgabe zu erzeugen, falls diese fehlschlägt.

{ReqOptFunc3} Das Logging der Zustände während des Programmablauf kann mit der Library Easylogging++ erfolgen

{ReqOptFunc4} Das Logging der Zustände während des Programmablauf kann mit der Library Boost Logging erfolgen

#### Formatieren der Ausgabe

Die Formatierung der Ausgabe ist kein Bestandteil des Programmentwurfs. Daher sollte keine Zeit in die Formatierung des Ergebnisses verwendet werden. Um aber eine besser Übersicht im Arbeitsergebnis zu erhalten wäre eine Formatierung der Header- und Source Datei vorteilhaft. Hierbei bietet sich astyle an.

{ReqOptFunc5} Das Formatieren des erzeugten Codes kann mit Artistic Style erfolgen

#### Testen von Teilfunktionalitäten

Um die einzelnen Programmteile auf Funktionalität zu untersuchen ist es angeraten einen Unittest für die einzelnen Programmteile zu erstellen. Es hat sich aber in der Vergangenheit gezeigt, das sowohl die Studenten wie auch die Dozenten aus den Unittest nachvollziehen konnten wie das Programm funktioniert und häufig auch Fehler gefunden wurden. Zwei Libraries sind hier von Bedeutung, einmal die einfach zu implemntierende Catch2 die nur als reine Header Datei includiert werden kann sowie das etwas aufwendigere Framework von Boost Test. Der Vorteil von Catch2 ist die leichtere Implementierbarkeit, der Vorteil von Boost Test der bessere Funktionsumfang und die bessere Integrierbarkeit von C++.

{ReqOptFunc6} Das Testen der Module kann mit der Library Catch2 erfolgen

{ReqOptFunc7} Das Testen der Module kann mit der Library Boost Test erfolgen

## 2.4 Programmierung

{ReqOptFunc8} Das verwenden des ? Operators ist gestattet

{ReqOptFunc9} Bei einer Kombination aus if/else if das kein else hat kann das else mit einem leeren Codeblock mit entsprechender Kommentierung hinzugefügt werden

{ReqOptFunc10} Das verwenden von mehrdimensionalen Arrays soll vermieden werden

{ReqOptFunc11} Soweit technisch möglich sollten die Header- und Source Dateien in utf-8 Zeichencodierung erstellt werden

## 2.5 Programmbeispiele

### 2.5.1 Formatierung unter C++

Sehr präzise kann Text unter C formatiert werden. Im Prinzip gibt es diese Optionen auch unter C++. Ein einfaches Mittel die C Formatierung von printf auch unter C++ zu übernehmen ist die Template Klasse von Boost Format. Der Syntax ist identisch zu printf und es gibt auch Erweiterungen.

```
//https://www.boost.org/doc/libs/1_75_0/libs/format/doc/format.html
#include <iostream>
using namespace std;

#include <boost/format.hpp>
using namespace boost;

int main() {
    cout << format("%-20s %s") % "Hello" % "World"  << endl;
    return 0;
}
```

### 2.5.2 SAX mit Xerces-C

Die Xerces-C Library beinhaltet alles was ein guter XML Parser kann, SAX, Dome, X-Path etc. Hier ein Beispiel mit SAX das ein gültiges XML Dokument parsen kann.

```
//g++ -o sax.exe sax.cpp -IC:\~PathToLib~\xerces-c\include -LC:\~PathToLib
    ~\xerces-c\lib -lxerces-c.dll -std=c++20
//Aufruf: sax datei.xml
#include <xercesc/util/XMLString.hpp>
#include <xercesc/parsers/SAXParser.hpp>
#include <xercesc/sax/HandlerBase.hpp>
#include <xercesc/util/OutOfMemoryException.hpp>
XERCES_CPP_NAMESPACE_USE

#include <locale>
```

```
10 #include <codecvt>
   #include <string>
12 #include <iostream>
   using namespace std;
14
   class SimpleSAXParser : public HandlerBase {
16 public:
       virtual void startDocument() {
18         cout << "Dokument beginnt!" << endl;
       }
20
       virtual void endDocument() {
22         cout << "Dokument ist zu Ende!" << endl;
       }
24
       virtual void startElement(const XMLCh* const name, AttributeList&
       attributes) {
26         cout << "Start-Element: " << converter.to_bytes(name) << endl;

28         for (XMLSize_t i = 0; i < attributes.getLength(); ++i)
               cout << converter.to_bytes(attributes.getName(i)) << '=' <<
       converter.to_bytes(attributes.getValue(i)) << endl;
30     }

32     virtual void endElement(const XMLCh* const name) {
           cout << "Element ist zu Ende: " << converter.to_bytes(name) << endl
       ;
34     }

36     virtual void characters(const XMLCh* const chars, const XMLSize_t
       length) {
           cout << "Buchstaben (" << length << "):" << converter.to_bytes(
       chars, chars + length) << endl;
38     }
   private:
40     wstring_convert<codecvt_utf8_utf16<char16_t>,char16_t> converter;
   };
42
   int main(int argc, char* argv[]) {
44     try {
           XMLPlatformUtils::Initialize();
46     }
       catch (const XMLException& toCatch) {
48         char* message = XMLString::transcode(toCatch.getMessage());
           cerr << "Error during initialization! :\n"
50              << message << "\n";
           XMLString::release(&message);
52         return 1;
       }
54
       SAXParser* parser = {nullptr};
56     parser = new SAXParser;

58     int errorCount = {0};
```

```
60      try
        {
62          //Das eigentliche Parsen der Datei
            SimpleSAXParser handler;
64          parser->setDocumentHandler(&handler);
            parser->parse(argv[1]);
66          errorCount = parser->getErrorCount();
        }
68      catch (const OutOfMemoryException&)
        {
70          XERCES_STD_QUALIFIER cerr << "OutOfMemoryException" <<
        XERCES_STD_QUALIFIER endl;
        }
72      catch (const XMLException& toCatch)
        {
74          char* message = XMLString::transcode(toCatch.getMessage());

76          //XMLString::release(message);
            /*
78          XERCES_STD_QUALIFIER cerr << "\nAn error occurred\n  Error: "
                                      << StrX(toCatch.getMessage())
80                                    << "\n" << XERCES_STD_QUALIFIER endl;
                                      */
82          cerr << "XMLException: " << message << endl;
        }
84      catch(...) {
            cerr << "Unbekannter Fehler" << endl;
86      }

88      cout << "Anzahl Fehler: " << errorCount << endl;

90      //Parser sauber beenden
        delete parser;
92      //Terminate muss immer am Schluss stehen
        XMLPlatformUtils::Terminate();
94
        return 0;
96  }
```

### 2.5.3 XML, utf-16, utf-8 (ASCII)

XML Parser arbeiten mit utf-16. Die Weiterverarbeitung in utf-16 ist sperrig. Eine Wandlung in utf-8 bietet sich hier an. Da im Programmentwurf nur mit ASCII gearbeitet wird ist utf-8 das richtige Format, denn ASCII sind die ersten 127 Zeichen von utf-8. Anbei ein paar Beispiele wie sich die utf-16 in utf-8 wandeln lässt.

```
#include <xercesc/util/XMLString.hpp>
2 const XMLCh * const UTF16TEXT = {u''Ich bin ein Text im utf-16 Format''};
```

```
//Eine Wandlung in utf-8 geht mit der statischen Klasse XMLString::
    transcode
const char *utf-8 = XMLString::transcode(UTF16TEXT);
cout << utf-8 << endl;
XMLString::release(&utf-8);

//Aufwendig ist das freigeben des Speichers unter transcode.
//Die STL bietet hier eine einfachere Methode
wstring_convert<codecvt_utf8_utf16<char16_t>, char16_t> convUTF16UTF8;

//Falls sich die strlen mit der Methode XMLString::stringLen ermittel
    laesst
string tagvalue = convUTF16UTF8->to_bytes(chars);
//Falls eine Laengenangabe mit verarbeitet werden muss
string tagvalue = convUTF16UTF8->to_bytes(chars,chars + length);
```

### 2.5.4 Muster XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<GetOptSetup SignPerLine="79">
  <Author Name="Thomas Staudacher, Andreas Maus" Phone="07541-77-961003"
    Mail="thomas.staudacher@zf.com, andreas.maus@zf.com" />
  <HeaderFileName>options.h</HeaderFileName>
  <SourceFileName>options.cpp</SourceFileName>
  <NameSpace>DHBW</NameSpace>
  <ClassName>COptionParser</ClassName>
  <OverAllDescription>
    <Block>Erstellt einen Rumpf zum einlesen von Argumente aus der
    Kommandozeile.</Block>
    <Block>Es kann sowohl mit innenliegenden Container wie externer
    Klassenanbindung eine Datenhaltung erfolgen.</Block>
    <Block>Sobald ein Methodenaufruf abstrakt ist, wird die Basisklasse
    abstrakt.</Block>
    <Block>Fuer die Formatierung der generierten Dateien wird astyle
    verwendet.</Block>
  </OverAllDescription>
  <SampleUsage>
    <Sample>getoptgen [options] ... QUELLE</Sample>
    <Sample>getoptgen [--out-path] ... QUELLE</Sample>
  </SampleUsage>
  <Options>
    <!-- Option help greift auf die interne Klasseninterne Methode
    printHelp zu. Ein gleichzeitiger Aufruf mit version und parse-only ist
    nicht erlaubt. -->
    <Option Ref="1" ShortOpt="h" LongOpt="help" Exclusion="2,3"
    ConnectToInternalMethod="printHelp" Description="Diese Hilfe ausgeben
    und beenden" />
    <!-- Option version greift auf die interne Klassenexterne (abstrakte)
    Methode printVersion zu. Ein gleichzeitiger Aufruf mit helpund parse-
    only  ist nicht erlaubt. -->
```

```
22      <Option Ref="2" ShortOpt="v" Interface="Version" Exclusion="1,3"
        ConnectToInternalMethod="printVersion" Description="Gibt die Version
        des Programms aus und beendet" />
        <!-- Option out-path braucht ein zusätzliches Argument und schreibt auf
         einen Klassenintern generierten String das Argument. Die Option help,
        version und parse-only  darf nicht angegeben sein -->
24      <Option LongOpt="out-path" HasArguments="Required" Exclusion="1,2,3"
        Interface="OutputPath" Description="Der Pfad wo das Ergebnis
        hingenriert werden soll (sonst ins aktuelle Verzeichnis)" />
        <Option LongOpt="astyle-path" HasArguments="Required" Interface="
        AstylePath" Exclusion="1,2,3" Description="Der Pfad wo die Astyle
        executable gefunden werden kann" />
26      <!-- Option sign-per-line kann ein zusätzliches Argument übergeben
        werden und schreibt auf einen Klassenintern generierten Integer
        SignPerLine. Wenn kein Parameter übergeben wird ist der Defaultwert 79.
         Die Option help, version und parse-only darf nicht angegeben sein -->
        <Option LongOpt="sign-per-line" HasArguments="optional" Interface="
        SignPerLine" Exclusion="1,2,3" ConvertTo="Integer" DefaultValue="79"
        Description="Die Anzahl der Zeichen pro Linie für den Helptext. Ohne
        Argument wird der Standartwert genommen." />
28      <Option ShortOpt="n" LongOpt="only-if-newer" Interface="OnlyIfNewer"
        Exclusion="1,2,3" Description="Generiert nur wenn die Eingangsdatei
        neuer ist wie die bereits generierte" />
        <Option LongOpt="no-format" Interface="NoFormat" Exclusion="1,2,3"
        Description="Erzeugte Datei wird nicht formatiert" />
30      <!-- Option parse-only liesst ein zusätzliches Argument ein und ruft
        die interne Klassenexterne (abstrakte) Methode ParseXML auf. Die Option
         help, version darf nicht angegeben sein -->
        <Option Ref="3" LongOpt="parse-only" HasArguments="Required"
        ConnectToInternalMethod="ParseXML" Exclusion="1,2" Description="Parst
        die Datei einmal und beendet das Programm" />
32    </Options>
  </GetOptSetup>
```