Projektarbeit:
“htmlLaTeX” - Konvertierungsoftware
–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

2 Listings

2.1 cconsole

```
#include "cconsole.h"

/** This class performs the format conversion via console input and output.
  * @author Bjoern Kaiser
  */
CConsole::CConsole(int argc, char* argv[])
{
    performInitialOperations(argc, argv);
};
CConsole::~CConsole()
{
};
void CConsole::performInitialOperations(int argc, char* argv[])
{
    /* 0 = executable's name
       1 = source file path
       2 = source file type
       3 = target file path
       4 = target file type
       5 = "-g" or "--gui"
    */
    if (argc == 5)
    {
        QString sourcefilepath(argv[1]);
        QString filetypestring(argv[2]);
        QFile file(sourcefilepath);
        CNode* root = 0;
        if (file.exists())
        {
            CDocumentData::FileType filetype = CDocumentData::Unknown;
            if (filetypestring.toLower() == "javadoc")
                filetype = CDocumentData::JavaDocHTML;
            CDocumentReader* reader = new CDocumentReader;
            root = reader->read(sourcefilepath, filetype);
            delete reader;
            std::cout << tr("Read source file(s) --> success").toStdString() << std::endl;
        }
        else
        {
            file.close();
            std::cout << tr("I/O error - the source file doesn't exit: ").toStdString() << ↘
                 ↪sourcefilepath.toStdString() << std::endl;
            return;
        }
        file.close();
        QString targetfilepath(argv[3]);
        filetypestring = QString(argv[4]);
        file.setFileName(targetfilepath);
        if (file.open(QFile::WriteOnly))
        {
            CDocumentData::FileType filetype = CDocumentData::Unknown;
            if (filetypestring.toLower() == "tex")
                filetype = CDocumentData::Tex;
            // root - converting...
            std::cout << tr("Perform conversion --> success").toStdString() << std::endl;
        }
        else
            std::cout << tr("I/O error - can't write to file: ").toStdString() << ↘
                 →targetfilepath.toStdString() << std::endl;
    }
    else
        std::cout << tr("Error - unexpected number of arguments.").toStdString() << std::endl;
};
```

Listing 1: Qt-Input-capiton

2.2 cdocumentdata

```
#include "cdocumentdata.h"
#include <iostream>
#include "constants.h"

/** This class holds information of one document (i.e. webpage). The information
  * consists of the file path (URL), a reference pointing to the corresponding node
  * in the document tree, which represents the whole document. This also stores a
  * QFileInfo-object for the whole documents' index document. The appropriate
  * document preprocessing is chosen by means of distinguished file types.
  * @author Bjoern Kaiser
  */
CDocumentData::CDocumentData(QFileInfo fileinfo, CNode* node, FileType filetype)
{
    this->_fileInfo = fileinfo;
    this->_node = node;
    this->_text = "";
    this->_fileType = filetype;
    this->_preprocessed = false;
};
/** Returns the URL to this document.
  * @author Bjoern Kaiser
  */
QFileInfo CDocumentData::fileInfo() const
{
    return this->_fileInfo;
};
/** Returns the pointer to the corresponding tree node of the whole document.
  * @author Bjoern Kaiser
  */
CNode* CDocumentData::node() const
{
    return this->_node;
};
/** Returns the document as a preprocessed QString.
  * @author Bjoern Kaiser
  */
QString CDocumentData::text()
{
    // the preprocessed document is stored in the attribute "_text"
    if (!_preprocessed)
        preprocessingHook();
    return _text;
};
/** This hook-method calls all preprocessing methods.
```

```
  * @author Bjoern Kaiser
  */
void CDocumentData::preprocessingHook()
{
    preprocessHTML();
};
/** This method changes the specified HTML-file in order to gain well-formed XML (XHTML).
  * @author Bjoern Kaiser
  */
void CDocumentData::preprocessHTML()
{
    if (_fileType != CDocumentData::JavaDocHTML)
        return;
    QString path = _fileInfo.filePath();
    if (DEBUG)
    {
        std::cerr << "Path: " << path.toStdString() << std::endl;
    }
    QFile file(path);
    if (!file.open(QFile::ReadOnly | QFile::Text))
        return;
    _text = QString(file.readAll()).toLatin1();
    file.close();
    // processing document
    // add missing finalizing slashes to empty elements
    // the following two lines create an array of QString objects
    QStringList elements;
    elements << "br" << "img" << "hr" << "meta" << "link";
    QRegExp regex;
    foreach (QString element, elements)
    {
        regex = QRegExp("<" + element + // opening tag and tag name
                        "([^\\/>])*" // consume all characters except '/' and '>'
                        "(?!\\/)>"); // the character '/' is missing before '>'
        // the empty element stored in the iteration variable 'element' is found without a '/'
        while(regex.indexIn(_text, 0) >= 0)
        {
            // insert missing '/'
            // regex.indexIn() returns the found position and
            // regex.cap(0).count() is the number of characters of mathed string
            _text.insert(regex.indexIn(_text, 0) + regex.cap(0).count() - 1, "/");
        }
    }
    _preprocessed = true;
};
```

Listing 2: Qt-Input-capiton

2.3 cdocumentreader

```
#include "cdocumentreader.h"
#include "constants.h"

/** This class creates a tree representation of a given document.
  @author Bjoern Kaiser
  */
CDocumentReader::CDocumentReader()
{
    _fileType = CDocumentData::Unknown;
```

```
};
/** This method
  * @param filetype contains the file filter string selected previously.
  * @author Bjoern Kaiser
  */
CNode* CDocumentReader::read(QString indexfilepath, CDocumentData::FileType filetype)
{
    _fileType = filetype;
    _indexFileInfo = QFileInfo(indexfilepath);
    // start reading the whole document tree
    CNode* root = new CNode(0, "html", 0);
    // add the index document to the stack of documents
    _documentStack.push(new CDocumentData(_indexFileInfo, root, _fileType));
    // begin processing the documents stored on the document stack
    while(!_documentStack.isEmpty())
    {
        CDocumentData* documentdata = _documentStack.pop();
        QDomDocument doc;
        QString errorStr = "";
        int errorLine = -1;
        int errorColumn = -1;
        if (doc.setContent(documentdata->text().toLatin1(), false, &errorStr, &errorLine, ↘
             →&errorColumn))
        {
            if (doc.documentElement().tagName().toLower() == "html")
                readElement(doc.documentElement(), documentdata->node());
            else
            {
                std::cerr << "Error in CDocumentReader::read()"
                        << std::endl << "\tat \"if (doc.setContent())\" returned false;"
                        << std::endl << "\tFile name: " << ↘
                             →documentdata->fileInfo().filePath().toStdString()
                        << std::endl << "\tError message: <html>-tag not found"
                        << std::endl;
            }
        }
        else
        {
            std::cerr << "Error in CDocumentReader::read()"
                    << std::endl << "\tat doc.setContent() returned false;"
                    << std::endl << "\tFile name: " << ↘
                         →documentdata->fileInfo().filePath().toStdString()
                    << std::endl << "\tError message: "
                    << std::endl << errorStr.toStdString() << " line="
                    << std::endl << QString::number(errorLine).toStdString()
                    << std::endl << "\tColumn=" << QString::number(errorColumn).toStdString()
                    << std::endl;
        }
        delete documentdata;
    }
    return root;
};
void CDocumentReader::readElement(QDomElement element, CNode* node)
{
    for (int i = 0; i < element.childNodes().count(); i++)
    {
        if (element.childNodes().at(i).nodeName().toLower() == "#text")
            node->setContent(element.childNodes().at(i).nodeValue());
        else
        {
            CNode* new_node = new CNode(node, element.childNodes().at(i).nodeName().toLower(), ↘
                 →node->layer() + 1);
            node->addChild(new_node);
            QDomNamedNodeMap attributes = element.childNodes().at(i).attributes();
            if (element.childNodes().at(i).nodeName().toLower() == "font")
                new_node->addAttribute("size", attributes.namedItem("size").nodeValue());
            else if (element.childNodes().at(i).nodeName().toLower() == "td")
            {
                new_node->addAttribute("align", attributes.namedItem("align").nodeValue());
                new_node->addAttribute("valign", attributes.namedItem("valign").nodeValue());
                new_node->addAttribute("width", attributes.namedItem("width").nodeValue());
            }
            else if (element.childNodes().at(i).nodeName().toLower() == "tr")
                new_node->addAttribute("bgcolor", attributes.namedItem("bgcolor").nodeValue());
            else if (element.childNodes().at(i).nodeName().toLower() == "th")
            {
                new_node->addAttribute("align", attributes.namedItem("align").nodeValue());
                new_node->addAttribute("colspan", attributes.namedItem("colspan").nodeValue());
            }
            else if (element.childNodes().at(i).nodeName().toLower() == "a")
            {
                if (DEBUG)
                {
                    std::cerr << "#\tReadElement - 'a': \n#\t\tindexFileInfo.absPath: " << ↘
                         →_indexFileInfo.absolutePath().toStdString()
                           << std::endl << "#\t\thref: " << ↘
                                →new_node->attributes()["href"].toStdString() << std::endl;
                }
                new_node->addAttribute("href", attributes.namedItem("href").nodeValue());
                // compose absolute file path
                QFileInfo myfileinfo;
                if (QFileInfo(_indexFileInfo.absolutePath() + new_node->attributes()["href"]).exists())
                    // unix
                    myfileinfo = QFileInfo(_indexFileInfo.absolutePath() + ↘
                         →new_node->attributes()["href"]);
                else
                    // windows
                    myfileinfo = QFileInfo(_indexFileInfo.absolutePath() + QDir::separator() + ↘
                         →new_node->attributes()["href"]);
                _documentStack.push(new CDocumentData(myfileinfo, new_node, _fileType));
               if(DEBUG)
                {
                   std::cerr << "# CDocumentReader::readElement()"
                           << std::endl << "#\tat \"if ↘
                                →(element.childNodes().at(i).nodeName().toLower() == \"a\")\" ↘
                                →returned true"
                           << std::endl << "#\tfound subdocument href=" << ↘
                                →new_node->attributes()["href"].toStdString() << std::endl;
                }
            }
            QDomElement new_element = element.childNodes().at(i).toElement();
            readElement(new_element, new_node);
        }
    }
};
```

Listing 3: Qt-Input-capiton

2.4 citemdelegate

```
#include "citemdelegate.h"
```

```
CItemDelegate::CItemDelegate(CModel* model, QObject* parent)
    : QItemDelegate(parent)
{
    this->model = model;
    this->colorFocusLine = QColor(0, 255, 255, 255);
    this->colorFocusBackground = QColor(180, 180, 180, 255);
    this->colorMarked = QColor(200, 0, 0, 255);
    this->color = QColor(0, 0, 0, 255);
    this->colorLayer0 = QColor(75,86, 32, 255);
    this->colorLayer1 = QColor(0, 0, 255, 255);
};
void CItemDelegate::setColorFocusLine(QColor color)
{
    this->colorFocusLine = color;
};
QColor CItemDelegate::getColorFocusLine() const
{
    return this->colorFocusLine;
};
void CItemDelegate::setColorFocusBackground(QColor color)
{
    this->colorFocusBackground = color;
};
QColor CItemDelegate::getColorFocusBackground() const
{
    return this->colorFocusBackground;
};
void CItemDelegate::setColorMarked(QColor color)
{
    this->colorMarked = color;
};
QColor CItemDelegate::getColorMarked() const
{
    return this->colorMarked;
};
void CItemDelegate::setColor(QColor color)
{
    this->color = color;
};
QColor CItemDelegate::getColor() const
{
    return this->color;
};
void CItemDelegate::setColorLayer0(QColor color)
{
    this->colorLayer0 = color;
};
QColor CItemDelegate::getColorLayer0() const
{
    return this->colorLayer0;
};
void CItemDelegate::setColorLayer1(QColor color)
{
    this->colorLayer1 = color;
};
QColor CItemDelegate::getColorLayer1() const
{
    return this->colorLayer1;
};
void CItemDelegate::paint(QPainter* painter, const QStyleOptionViewItem &option,
           const QModelIndex &index) const
{
    painter->save();
    QStyleOptionViewItem opt = option;
    opt.displayAlignment = Qt::AlignLeft | Qt::AlignVCenter;
    CNode* node = model->nodeFromIndex(index);
    QString text = node->name() + " (" + QString::number(node->layer()) + ")";
    painter->setPen(color);
    if (node->content() != "")
        text += ": " + node->content();
    // an item of the treeview is selected
    if (option.state & QStyle::State_Selected)
    {
        painter->save();
        painter->setBrush(colorFocusBackground);
        painter->fillRect(QRect(opt.rect.x(), opt.rect.y(), opt.rect.width() - 1, opt.rect.height() ↘
            →- 1), Qt::SolidPattern);
        painter->setPen(QPen(QBrush(colorFocusLine, Qt::SolidPattern), 1.0, Qt::DashLine, ↘
            →Qt::RoundCap, Qt::BevelJoin));
        painter->drawRect(QRect(opt.rect.x(), opt.rect.y(), opt.rect.width() - 1, opt.rect.height() ↘
            →- 1));
        painter->restore();
    }
    if ((node->layer() % 2) == 0)
        painter->setPen(colorLayer0);
    else if ((node->layer() % 2) == 1)
        painter->setPen(colorLayer1);
    // determine icon size
    //int size = opt.rect.height() - 2;
    //painter->drawText(QRect(opt.rect.x() + size + 4, opt.rect.y(), opt.rect.width(), ↘
        →opt.rect.height()), text, QTextOption(Qt::AlignLeft|Qt::AlignVCenter));
    painter->drawText(QRect(opt.rect.x(), opt.rect.y(), opt.rect.width(), opt.rect.height()), text, ↘
        →QTextOption(Qt::AlignLeft|Qt::AlignVCenter));
    painter->restore();
};
```

Listing 4: Qt-Input-capiton

2.5 cmodel

```
#include "cmodel.h"

CModel::CModel(QObject* parent) : QAbstractItemModel(parent), _root(0)
{
};
CModel::~CModel()
{
    delete _root;
};
void CModel::setRootNode(CNode* node)
{
    delete _root;
    _root = node;
    // reset() notifies the views to refetch data for visible items
    reset();
};
QModelIndex CModel::index(int row, int column, const QModelIndex &parent) const
{
    if (!_root
        || (row < 0)
        || (column < 0))
        return QModelIndex();
    CNode* parentNode = nodeFromIndex(parent);
    CNode* childNode = parentNode->childAt(row);
    if (!childNode)
        return QModelIndex();
    return createIndex(row, column, childNode);
};
CNode* CModel::nodeFromIndex(const QModelIndex &index) const
{
    // cast the index's void* to a CNode*
    if (index.isValid())
        return static_cast<CNode*>(index.internalPointer());
    // In a model the root node is represented by an invalid index.
    return _root;
};
int CModel::rowCount(const QModelIndex &parent) const
{
    if (parent.column() > 0)
        return 0;
    CNode *parentNode = nodeFromIndex(parent);
    if (!parentNode)
        return 0;
    return parentNode->count();
};
int CModel::columnCount(const QModelIndex &parent) const
{
    // just one column meets our needs
    return 1;
};
QModelIndex CModel::parent(const QModelIndex &child) const
{
    CNode* node = nodeFromIndex(child);
    if (!node)
        return QModelIndex();
    CNode* parentNode = node->parent();
    if (!parentNode)
        return QModelIndex();
    CNode* grandparentNode = parentNode->parent();
    if (!grandparentNode)
        return QModelIndex();
    int row = grandparentNode->indexOf(parentNode);
    return createIndex(row, 0, parentNode);
};
QVariant CModel::data(const QModelIndex &index, int role) const
{
    if (role != Qt::DisplayRole)
        return QVariant();
    CNode* node = nodeFromIndex(index);
    if (!node)
        return QVariant();
    return node->name() + ": " + node->content();
};
QVariant CModel::headerData(int section, Qt::Orientation orientation,
                            int role) const
{
    return QVariant();
};
CNode* CModel::root() const
{
    return this->_root;
};
```

Listing 5: Qt-Input-capiton

2.6 cnode

```
#include "cnode.h"

/**
 * Constructs a new object.
 * @param parent Initialiazes the object with the given CNode-object as its parent.
 * @author Bjoern Kaiser
 */
CNode::CNode(CNode* parent, QString name, qint64 layer) : _parent(parent),
    _layer(layer), _name(name), _content("")
{
    instCount++;
    this->_id = instCount;
    this->children = QList<CNode*>();
    this->_attributes = QMap<QString, QString>();
};
/**
 * This destructor deletes all child nodes by means of Qt's generic algorithms.
 * @author Bjoern Kaiser
 */
CNode::~CNode()
{
    qDeleteAll(children);
};
/**
 * This class instance counter is used to assign unique ID-numbers to each class instance.
 * @author Bjoern Kaiser
 */
qint64 CNode::instCount = 0;
/**
 * Returns the node's unique ID.
 * @author Bjoern Kaiser
 */
qint64 CNode::ID() const
{
    return this->_id;
};
/**
  * Returns the layer number of the node. The layer number is the distance to the
  * root node of the tree.
  * @author Bjoern Kaiser
  */
qint64 CNode::layer() const
{
    return this->_layer;
};
/**
 * Returns the node's name.
 * @author Bjoern Kaiser
 */
QString CNode::name() const
{
    return this->_name;
};
/**
 * Returns the node's text content.
 * @author Bjoern Kaiser
 */
QString CNode::content() const
{
    return this->_content;
```

```
};
/**
 * Returns node's attributes (such as size, width, align etc.).
 * @author Bjoern Kaiser
 */
QMap<QString, QString> CNode::attributes() const
{
    return this->_attributes;
};
/**
 * Sets the node's name.
 * @author Bjoern Kaiser
 */
void CNode::setName(QString name)
{
    this->_name = name;
};
/**
 * Sets the node's content.
 * @author Bjoern Kaiser
 */
void CNode::setContent(QString content)
{
    this->_content = content;
};
/**
 * Adds an attribute by means of a key-value-pair.
 * @author Bjoern Kaiser
 */
void CNode::addAttribute(QString key, QString value)
{
    this->_attributes[key] = value;
};
/**
 * Returns pointer to the child node with the specified index.
 * @author Bjoern Kaiser
 */
CNode* CNode::childAt(int index) const
{
    return children.value(index);
};
/**
 * Returns the number of direct child nodes.
 * @author Bjoern Kaiser
 */
int CNode::count() const
{
    return children.count();
};
/**
 * Sets the parent node.
 * @author Bjoern Kaiser
 */
void CNode::setParent(CNode* parent)
{
    this->_parent = parent;
};
/**
 * Returns the parent node.
 * @author Bjoern Kaiser
 */
CNode* CNode::parent()
{
    return this->_parent;
```

```
};
/**
 * Returns the index of the specified child node.
 * @author Bjoern Kaiser
 */
int CNode::indexOf(CNode* node) const
{
    return children.indexOf(node);
};
/**
 * Adds the given node collection to the node's child collection.
 * @param nodes is a collection of nodes.
 * @author Bjoern Kaiser
 */
void CNode::addChildren(QList<CNode*> nodes)
{
    for (int i = 0; i < nodes.count(); i++)
    {
        if (!containsChild(nodes.at(i)))
        {
            nodes.at(i)->setParent(this);
            children.append(nodes.at(i));
        }
    }
};
/**
 * Adds the given node to the node's child collection.
 * @author Bjoern Kaiser
 */
void CNode::addChild(CNode* node)
{
    if (node != 0)
    {
        if (!containsChild(node))
        {
            node->setParent(this);
            children.append(node);
        }
    }
};
/**
 * Checks if the node's child collection contains the specified node.
 * @author Bjoern Kaiser
 */
bool CNode::containsChild(CNode* node) const
{
    bool result = false;
    for (int i = 0; i < count(); i++)
    {
        if (children.at(i)->_id == node->_id)
        {
            result = true;
            break;
        }
    }
    return result;
};
/**
 * Removes the given node from the node's child collection.
 * @author Bjoern Kaiser
 */
void CNode::removeChild(CNode* node)
{
    if (children.removeOne(node))
```

```
        delete node;
};
```

Listing 6: Qt-Input-capiton

2.7 ctranslationdata

```
#include "ctranslationdata.h"

CTranslationData::CTranslationData() : _from(""), _to("")
{
    this->_requires = QList<CTranslationDataNode>();
};
QString CTranslationData::from() const
{
    return this->_from;
};
void CTranslationData::setFrom(QString from)
{
    this->_from = from;
};
QString CTranslationData::to() const
{
    return this->_to;
};
void CTranslationData::setTo(QString to)
{
    this->_to = to;
};
QList<CTranslationDataNode> CTranslationData::requires() const
{
    return this->_requires;
};
void CTranslationData::addRequiresNode(CTranslationDataNode requiresnode)
{
    this->_requires.append(requiresnode);
};
```

Listing 7: Qt-Input-capiton

2.8 ctranslationdatanode

```
#include "ctranslationdatanode.h"

CTranslationDataNode::CTranslationDataNode() : _name(""),
    _content("")
{
    this->_attributes = QMap<QString,QString>();
};
QString CTranslationDataNode::name() const
{
    return this->_name;
};
```

```
void CTranslationDataNode::setName(QString name)
{
    this->_name = name;
};
QString CTranslationDataNode::content() const
{
    return this->_content;
};
void CTranslationDataNode::setContent(QString content)
{
    this->_content = content;
};
QMap<QString,QString> CTranslationDataNode::attributes() const
{
    return this->_attributes;
};
void CTranslationDataNode::addAttribute(QString name, QString value)
{
    this->_attributes[name] = value;
};
```

Listing 8: Qt-Input-capiton

2.9 ctranslationmapper

```
#include "ctranslationmapper.h"
#include "constants.h"

CTranslationMapper::CTranslationMapper()
{
    this->_outputMap = QMap<QString,CTranslationData>();
};
void CTranslationMapper::createInputElementMap(QString inputfilepath)
{
};
void CTranslationMapper::createOutputElementMap(QString outputfilepath)
{
    _outputMap = QMap<QString,CTranslationData>();
    QFile file(outputfilepath);
    if (!file.open(QFile::ReadOnly | QFile::Text))
    {
        if(DEBUG)
        {
            std::cerr << "CTranslationMapper.createOutputElementMap(): file.open() returned ↘
                ↪false\n\tPath: " << outputfilepath.toStdString() << std::endl;
        }
        return;
    }
    // load content of the XML file into "filecontent"
    QString filecontent = QString(file.readAll()).toLatin1();
    file.close();
    QDomDocument doc;
    QString errorMsg = "";
    int errorLine = -1;
    int errorColumn = -1;
    if (doc.setContent(filecontent, &errorMsg, &errorLine, &errorColumn))
    {
        QDomElement root = doc.documentElement();
        // create a node list of "node"-elements
        QDomNodeList nodes = root.elementsByTagName("node");
        for (int i = 0; i < nodes.count(); i++)
        {
            // create a node list from "node"-element's child nodes
            QDomNodeList subnodes = nodes.at(i).childNodes();
            CTranslationData translationdata;
            for (int j = 0; j < subnodes.count(); j++)
            {
                if (subnodes.at(j).nodeName().toLower() == "from")
                    translationdata.setFrom(subnodes.at(j).nodeValue());
                else if (subnodes.at(j).nodeName().toLower() == "to")
                    translationdata.setTo(subnodes.at(j).nodeValue());
                else if (subnodes.at(j).nodeName().toLower() == "requires")
                {
                    QDomNodeList requiresnodes = subnodes.at(j).childNodes();
                    for (int k = 0; k < requiresnodes.count(); k++)
                    {
                        CTranslationDataNode datanode;
                        datanode.setName(requiresnodes.at(k).nodeName());
                        datanode.setContent(requiresnodes.at(k).nodeValue());
                        QDomNamedNodeMap attributes = requiresnodes.at(k).attributes();
                        for (int l = 0; l < attributes.count(); l++)
                            datanode.addAttribute(attributes.item(l).nodeName(), ↘
                                 →attributes.item(l).nodeValue());
                        translationdata.addRequiresNode(datanode);
                    }
                }
            }
            _outputMap[translationdata.from()] = translationdata;
        }
    }
    else if (DEBUG)
    {
        std::cerr << "CTranslationMapper.createOutputElementMap(): doc.setContent returned ↘
             →false\n\tFilecontent: " << filecontent.toStdString() << std::endl;
    }
};
```

Listing 9: Qt-Input-capiton

2.10 main

```
#include <QtGui/QApplication>
#include <QtCore/QCoreApplication>

#include "mainwindow.h"
#include "cconsole.h"
#include "constants.h"

bool DEBUG = true;
int main(int argc, char* argv[])
{
    /* 0 = executable's name
       1 = source file path
       2 = source file type
       3 = target file path
       4 = target file type
       5 = "-g" or "--gui"
    */
    // open the GUI if argument "-g" or "--gui" is given
    bool showgui = false;
    for (int i = 0; i < argc; i++)
    {
        if ((QString(argv[i]).toLower() == "-g")
            | (QString(argv[i]).toLower() == "--gui"))
        {
            showgui = true;
            break;
        }
    }
    if (showgui)
    {
        QApplication a(argc, argv);
        MainWindow w(argc, argv, 0);
        w.show();
        return a.exec();
    }
    else if (argc == 1)
        std::cerr << "usage: htmlatex INPUTFILE FORMAT OUTPUTFILE FORMAT [-g|--gui]\n"
                << " e.g. htmlatex index.html javadoc manual.tex tex -g\n"
                << "\t-g, --gui \tLaunch the GUI\n"
                << "\t-h, --help \tShow some examples\n\n"
                << "\tSee the \"README\" file for further information." << std::endl;
    else if (argc == 2)
    {
        if ((QString(argv[1]).toLower() == "-h")
            | (QString(argv[1]).toLower() == "--help"))
        {
            QString helpstring("Examples with GUI:\n\nopen file initially:\n\nhtmlatex index.html ↘
                →javadoc -g\nhtmlatex index.html --gui javadoc");
            helpstring += "\n\nconvert file initially:\n\nhtmlatex -g index.html javadoc ↘
                →mytexoutput.tex tex\nhtmlatex index.html javadoc --gui mytexoutput.tex tex";
            helpstring += "\n\nExample with console:\n\nconvert file:\n\nhtmlatex index.html ↘
                →javadoc mytexoutput.tex tex";
            std::cerr << helpstring.toStdString() << std::endl;
        }
    }
    else
    {
        QCoreApplication a(argc, argv);
        CConsole console(argc, argv);
        exit(0);
        return a.exec();
    }
};
```

Listing 10: Qt-Input-capiton

2.11 input-javadoc.xml

```
<?xml version="1.0" encoding="utf−8" ?>
<!DOCTYPE javadoc [
<!ELEMENT javadoc (element∗)>
<!ELEMENT element (name, content?, attribute∗)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT attribute (#PCDATA)>
]>
<javadoc>
    <element>
        <name>font</name>
        <attribute>size</attribute>
    </element>
    <element>
        <name>td</name>
        <attribute>align</attribute>
        <attribute>valign</attribute>
        <attribute>width</attribute>
    </element>
    <element>
        <name>tr</name>
        <attribute>bgcolor</attribute>
    </element>
    <element>
        <name>th</name>
        <attribute>align</attribute>
        <attribute>colspan</attribute>
    </element>
    <element>
        <name>a</name>
        <attribute>href</attribute>
    </element>
</javadoc>
```

Listing 11: XML-Input-capiton

2.12 output-tex.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE tex [
<!ELEMENT tex (node*)>
<!ELEMENT node ((from, to, requires?)*)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT requires (location*)>
<!ELEMENT location (#PCDATA)>

<!ATTLIST location position (start|content|end) "content">
]>

<tex>
<node>
  <from>
    <![CDATA[<table>INHALT</table>]]>
  </from>
  <to>
    begin{longtable}{|p{.2textwidth}|p{.8textwidth}|}
    hline
    INHALT
    end{longtable}
  </to>
  <requires>
    <location position="start">
      usepackage{longtable}
```

```
        </location>
      </requires>
    </node>
    <node>
      <from>
        <![CDATA[<h1>INHALT</h1>]]>
      </from>
      <to>
        \section{
        INHALT
        }
      </to>
    </node>
  </tex>
```

Listing 12: XML-Input-capiton

3 Funktionsumfang

3.1 Konsolenanwendung

3.2 Grafische Oberfläche

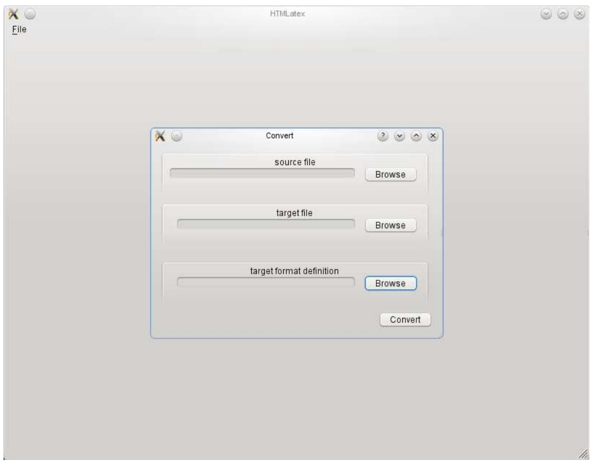
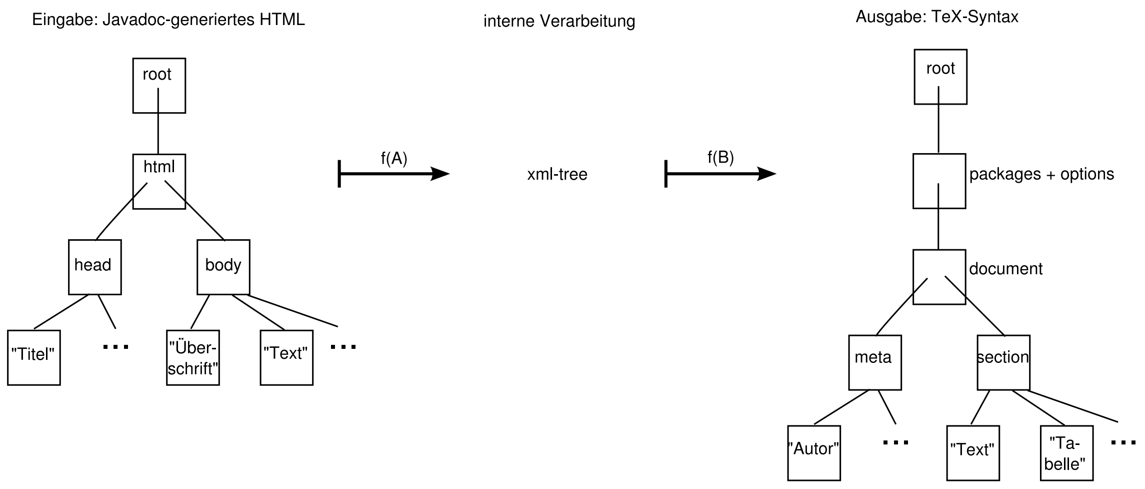


Abbildung 1: BILDUNTERSCHRIFT

4 Implementierung



5 Ausblick

6 Leistungsanforderungen

Keine besonderen Leistungsanforderungen.

7 Abgabe

Termin: Montag, 07.03.2011, Ort wird vom Betreuer bekannt gegeben.

- Projektordner mit CD
- kurze Präsentation des Ergebnisses

Literatur

[Lan06] Lang, Hans Werner: *Algorithmen in Java*. Oldenbourg, 2. Auflage, 2006.

[Sed92] Sedgewick, Robert: *Algorithmen in C++*. Addison-Wesley, 1. Auflage, 1992.

8 Erklärung

Hiermit erklären wir, dass das Projekt htmlLaTeXvon uns selbständig und ohne Hilfe Dritter erarbeitet und realisiert wurde.

Björn Kaiser

Björn Baß