Hochschule Heilbronn
Technik - Wirtschaft - Informatik
Heilbronn University

interdisziplinäres Institut
für intelligente
Geschäftsprozesse
Hochschule Heilbronn

Studienarbeit / Student research project

Developing a LaTeX plugin for JSPWiki

Steffen Schramm, 159978

July 31, 2006

Carried out at the i3G Institute, Heilbronn University.
Study course: Software Engineering
Referent: Professor Dr.-Ing. Tomas Benz
Tutor: Christoph Sauer

Abstract

This work compares several ways of creating printable documents: Typesetting systems like TeX, word processors and desktop publishing software. All of them have their right to exist, but they use an old-fashioned way of creating documents. Wikis, on the other hand, are web-based and offer many useful features - collaborative writing and version history are only two of them. The success of Wikis shows how the way of working together has changed. Wikis are simple and can be used by everyone. Information is available online and can be accessed from any computer. Documents can be split up in multiple pages and each part can be edited by different users at the same time.

A missing feature of most Wikis, however, is the ability to create printed documents. As Wikis are web-based, printing has to be done using the browser. But the quality of a printed web page is not acceptable - HTML was never made for printing. This problem is addressed by developing an extension for JSPWiki. It is realized as a plugin and makes use of the following technologies:

- The typesetting system LaTeX is used as intermediate format to create the PDF documents suitable for articles, reports or even books.
- The parser for the Wiki Markup is developed using the parser generator JavaCC.

This document was written in JSPWiki using the developed LaTeX plugin.

Contents

List of Figures

1 Introduction

1.1 Motivation

Wikis become more and more popular. An example for the success of the idea behind Wikis is the Wikipedia project[1], an online encyclopedia written mainly by its visitors. One reason for the success of Wikis is that they follow the KISS[2] principle: The simplest solution is normally the best. When using a Wiki, anyone can participate. Editing a web page is as fast and simple as posting to a web forum or writing an email. Pages are written in Wiki markup, which is simpler and faster to learn than HTML. The published information is available immediately and can be reviewed by others. It is no coincidence that *WikiWiki* means *quick* in the hawaiian language.

A Wiki engine is a collaborative platform suitable for lots of different tasks. Personal homepages are based on Wiki technology, because it supersedes editing and uploading HTML files by hand. Content management systems are replaced by Wikis, because Wikis are simpler to use. Companies use them for their intranet platform while individuals keep their notes, links and tasks in their personal Wikis. Companies can use Wiki engines instead of word processors to write their documents. This way the documents are available online and always up-to-date. The ability to split text several pages helps managing large documents while the page history mechanism allows to restore older versions and view differences between two versions.

1.2 The problem

Although Wikis are perfectly suited for collaborative writing of documents of any kind, they do not have an important feature, namely combining the content to a printable document. It is possible to write a whole book in a Wiki, but you won't be able to produce a result which is ready for press, unless you have a special tool to convert Wiki content to another, more printer-friendly output format.

As Wiki pages are basically just HTML[11] pages, printing has to be done using the browser. HTML however, was made for online reading and not for printing. Web pages are continous

[1] Wikipedia, http://www.wikipedia.org
[2] Keep it simple, stupid!

while printouts are split to fit on the paper. Page layout, used fonts and colors differ from the needs of printed documents. This can only be partly overcome by using a separate style sheet for printing. The World Wide Web Consortium (W3C) is addressing these problems with several new technologies that focus on printing [4]. All of them require to write documents in a special format.

Another challenge when printing Wiki content is the fact that information in Wikis is often distributed on many pages. Wikis can consist of hundreds or thousands of pages linking to each other. To print all information about a specific topic, each page has to be printed separately. In some Wiki engines this can be partially solved by creating a new page and using a page inclusion mechanism to insert all pages about the topic.

Other features are missing completely. Inserting a list of figures or making use of bibliographies are only some of them. Nowadays, Wikis are not only a system for taking notes, but already a replacement for content management systems. If all of the named problems were addressed, a Wiki could also be a serious alternative to word processors in many cases.

1.3 The goals of this study

The main goal is to provide a solution that makes it possible to use a Wiki as an document authoring system especially usable for articles, reports, documentation and books. The Wiki should be able to replace other software usable for this task while retaining its original strengths like ease of use and collaborative working.

By using the new solution, it should at least be possible to

- Convert a Wiki page to a high-quality printable document
- Combine multiple Wiki pages to one result document
- Make use of bibliographies and references

The chosen wiki engine is JSPWiki[3], as this is the Wiki engine used at the i3G Institut where this study is carried out. JSPWiki is a Wiki build upon the Java[**?**] and JSP[10] technology. It is extensible by supporting custom templates and plugins. Thus, the solution is written in Java and provided as a plugin for JSPWiki. The preferred output format is the Portable Document Format (PDF)[1]. This format has become the de-facto-standard for the electronic exchange of documents. Most important, PDF documents render and print the same on any platform. PDF viewers should also be available everywhere.

[3] JSPWiki, http://www.jspwiki.org

The rest of this document is structured as follows:

- Chapter 2 (”Related work”) discusses existing solutions to create printable documents
- Chapter 3 (”Technologies”) explains the most important technical details of the used technologies
- Chapter 4 (”Concept”) introduces the design and architecture of the plugin
- Chapter 5 (”Conclusion”) describes what has been reached and what can be improved
- Appendix A contains the plugin documentation.
- Appendix B contains installation notes

2 Related work

2.1 Document creating

There exist many programs that can be used to write articles and reports. This chapter gives an overview about the existing types of programs and their pros and cons. It will become clear that neither of these programs are a valid replacement for the solution that has been developed during this study.

2.1.1 Typesetting overview

To create a textual document, it has to be typeset. To prevent misconception, it is necessary to explain what is meant by *typesetting*.

- **Typesetting**: Typesetting was invented during the 15th century in Europe by Johannes Gutenberg. The term *typesetting* refers to the process of creating textual material using a frame consisting of resuable letters (also known as moveable type). Since the begin of the digital era typesetting is mainly done by computers.
- **Typesetting software**: Generally, any program useable to create textual documents can be seen as typesetting software. But then again, the term typesetting software rarely refers to word processors or desktop publishing tools. These two kinds of software will be explained further down in this chapter. In this study, typesetting-only tools like TeX are referred to as *plain typesetting* software. This is not an offical term, but it is needed to separate typesetting-only programs from word processors and desktop publishing software.

Plain typesetting tools, word processors and desktop publishing tools are three different kinds of software with different benefits and drawbacks. While they all can be used to create textual documents, they use different concepts and ways of accomplishing this. Which solution fits best depends on the type of the document.

2.1.2 Plain typesetting software

History of typesetting software

Typesetting software was invented more than four decades ago at a time when no personal computers existed. The oldest typesetting software was RUNOFF for the CTSS operating system written in 1964 by Jerome H. Saltzer[9]. It consisted of two programs, TYPSET to create the documents and RUNOFF to create the output. Successors of RUNOFF are roff and its variants. Groff[1] is still used today to create the manpages (”manual pages”) on UNIX and UNIX-like operating systems like MacOS and Linux.

The most well-known typesetting system is TeX, developed by Donald E. Knuth[6]. He wrote TeX because he was not satisfied with the typographic quality of his book series *The Art of Computer Programming*[5]. TeX is still known to produce very high quality output, especially when typesetting mathematical formulas. The TeX code has not changed since many years - except for bug fixing. To Knuth, a stable typesetting system is more important than having new features. A document written in TeX 15 years ago can still be printed in the same quality today. As TeX is open source and TeX documents are written in plain text, this will probably also be possible in 100 or more years.

Writing documents

Plain typesetting programs come without a graphical user interface. The input text is formatted using special markup, similar to HTML. TeX for example uses commands starting with a backslash. A mathematical formula in TeX looks like *-b\pm\sqrt{b^2-4ac} \over {2a}* and it is printed as $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$.

Using markup code instead of a WYSIWYG[2] program gives the user much more control about the resulting document. However, few people use TeX directly to create documents; most people use LaTeX instead. LaTeX is a large set of macros for TeX written by Leslie Lamport in 1984[7] and now maintained by the LaTeX project[3].

The idea behind LaTeX is that typesetting should be done by the computer. Most people have no special knowledge in typesetting and thus cannot answer simple formatting questions correctly.

Some of these questions are:

[1] GNU Groff, http://www.gnu.org/software/groff/
[2] What you see is what you get
[3] The LaTeX Project, http://www.latex-project.org/

- How large should the document title be?
- Should a section heading be formatted in bold?
- Should I underline a word or make use of italics?

Some people have learned typesetting and know what looks best, but most people don't. Lamport describes this problem in an article where he concludes that document formatting should be done in a logical way, not a visual one[8]. The document author only should mark some text as a heading, but not decide about the way headings are formatted. Letting the typesetting program decide how to format a text leads to better results.

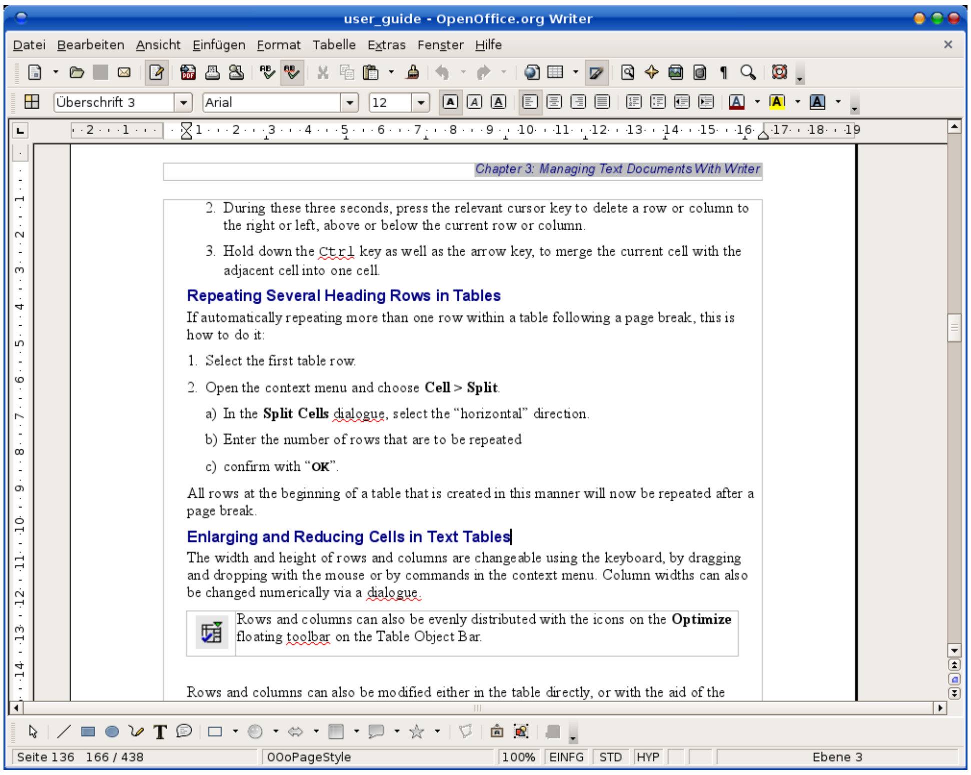
This method works best for documents that follow a special structure. LaTeX is especially useful to typeset articles, reports, books or presentations. By installing additional LaTeX packages it can even be used to create cooking recipes. But nobody would use LaTeX to create a poster for a film. Lamport admits in his article that automatic typesetting cannot be used for all kind of documents. The reason is that a typesetting system can only typeset a document when someone has created the necessary typesetting instructions for the given document type.

2.1.3 Word processing software

Word processing software is installed on almost every desktop computer. Popular representatives of this category are Microsoft Word[4] and OpenOffice.org Writer[5]. Word processors could be seen as some kind of electronic typewriters. Text formatting with different styles and sizes, including images and tables are the most used features of word processors. In addition, todays word processors come with features like spellchecking and automatic generation of a table of contents.

As the basic functions of word processors can be learned easily, they are both used at home and in companies. Writing a letter using a word processor is more intuitive than learning special typesetting commands like in TeX. Text formatting is done using a WYSIWYG approach by clicking a button or pressing a hotkey. The document author always sees a preview of the resulting document; all changes are reflected directly. Formatting is usually done in a visual way, not using the logical way Lamport prefers. However, logical formatting can also be done using word processors. Instead of changing the font size for a heading, it is possible to mark a text as a heading. The way headings are formatted can be defined separately and changed at any time.

[4] Microsoft Office, http://office.microsoft.com
[5] OpenOffice.org, http://www.openoffice.org


user_guide - OpenOffice.org Writer
Datei Bearbeiten Ansicht Einfügen Format Tabelle Extras Fenster Hilfe
Überschrift 3
Arial
12
Seite 136 166 / 438
OOoPageStyle
100%
EINFG
STD
HYP
Ebene 3

Chapter 3: Managing Text Documents With Writer

2. During these three seconds, press the relevant cursor key to delete a row or column to the right or left, above or below the current row or column.
3. Hold down the `Ctrl` key as well as the arrow key, to merge the current cell with the adjacent cell into one cell.

Repeating Several Heading Rows in Tables

If automatically repeating more than one row within a table following a page break, this is how to do it:

1. Select the first table row.
2. Open the context menu and choose **Cell** > **Split**.
   a) In the **Split Cells** dialogue, select the "horizontal" direction.
   b) Enter the number of rows that are to be repeated
   c) confirm with "**OK**".

All rows at the beginning of a table that is created in this manner will now be repeated after a page break.

Enlarging and Reducing Cells in Text Tables

The width and height of rows and columns are changeable using the keyboard, by dragging and dropping with the mouse or by commands in the context menu. Column widths can also be changed numerically via a dialogue.

Rows and columns can also be evenly distributed with the icons on the **Optimize** floating toolbar on the Table Object Bar.

Rows and columns can also be modified either in the table directly, or with the aid of the

Figure 2.1: Editing a document using OpenOffice.org Writer

More than typesetting?

Word processors are not just graphical typesetting systems. Often being part of a complete Office package, Word processors can be combined with spreadsheet and drawing programs or even databases. Tables from a spreadsheet can be inserted into a textual document and combined with a chart representing the table data in a graphical way. Documents can be batch-mailed to a list of clients listed in a database. Names and addresses are automatically filled into placeholders in the document. Word processors contain almost everything needed to write a document. They come with lots of features not included in ordinary typesetting systems like TeX, but they are not able to replace TeX completely. In fact, there are lots of people using TeX or LaTeX instead of word processors, and there are several reasons for this:

1. Missing support for mathematical formulas
2. Instability
3. Bad typographic quality

Support for mathematical formulas was very important to Knuth, so he made sure that TeX supports them very well. But there exist some solutions to typeset mathematical formulas in word processors, so this is not the real problem.

The second point is more important. Software cannot be proved to be bug-free. Even TeX, which is known to be stable since years, can contain bugs. Word processors are a very complex type of software, and new features are added with every release. None of the word processors is as stable as TeX is, and problems appear especially when writing large documents with lots of images, references and footnotes.

The most important reason is the bad typographic quality of word processors. At first, there seems to be no reason why a word processor should not be able to reach the same quality as TeX does. There is no magic behind TeX and it is open source too. Allin Cottrell, however, has found an explanation why TeX's typesetting algorithm is still superior to the algorithms of word processors[2]: While compiling TeX documents can take some time, word processors have to update the document layout in real time. Computers get faster and faster, but it seems that the complex algorithms of TeX take too long to be used in word processing software. This is a drawback of the WYSIWYG approach.

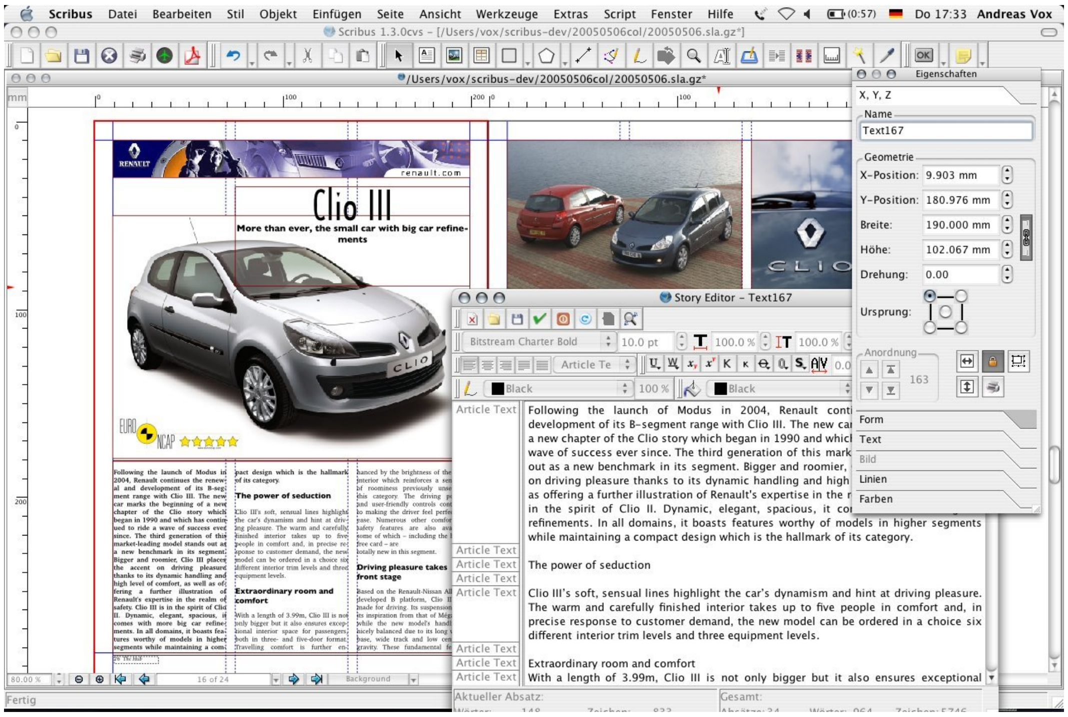
On the other hand, word processors provide many useful features not existing in ordinary typesetting systems. Both of them have their right to exist and the same is true for desktop publishing software.

2.1.4 Desktop publishing software

Desktop publishing (DTP) fills the gap between ordinary typesetting systems and word processors. Compared to word processors, DTP offers advanced layouts and is often used by professionals working at publishers or advertisting agencies. Flyers, brochures and posters are typically created with desktop publishing tools. Large books or multi-column articles can also be typeset using DTP. Popular desktop publishing tools are QuarkXPress and Adobe Framemaker. A free alternative is Scribus.

DTP originated from professional typesetting needs. Early word processors, offering only basic text formatting and simple layouts, could not fulfill the requirements of professional publishers. On the other hand, features like spellchecking and indexing were typical word processing features. Today, the differences begin to vanish. Word processors support complex layouts, and features like spellchecking are integrated into desktop publishing software.

Allin Cottrells theory seems to prove false here: DTP tools manage to create high quality output similar to TeX while using a WYSIWYG approach. Are the complex typesetting algorithms of TeX still fast enough for graphical software? The answer is: No. Desktop publishing tools do not use the same algorithms as TeX. Some DTP tools focus on large text documents includ-


Scribus Datei Bearbeiten Stil Objekt Einfügen Seite Ansicht Werkzeuge Extras Script Fenster Hilfe
(0:57) Do 17:33 Andreas Vox
Scribus 1.3.0cvs – [/Users/vox/scribus-dev/20050506col/20050506.sla.gz*]
/Users/vox/scribus-dev/20050506col/20050506.sla.gz*
Clio III
More than ever, the small car with big car refinements
renault.com
Story Editor – Text167
Eigenschaften
Fertig

Bitstream Charter Bold 10.0 pt 100.0 % 100.0 %
Article Te
Black 100 % Black

Article Text: Following the launch of Modus in 2004, Renault conti
development of its B–segment range with Clio III. The new car
a new chapter of the Clio story which began in 1990 and which
wave of success ever since. The third generation of this mark
out as a new benchmark in its segment. Bigger and roomier,
on driving pleasure thanks to its dynamic handling and high
as offering a further illustration of Renault's expertise in the r
in the spirit of Clio II. Dynamic, elegant, spacious, it co
refinements. In all domains, it boasts features worthy of models in higher segments while maintaining a compact design which is the hallmark of its category.

Article Text: The power of seduction

Article Text: Clio III's soft, sensual lines highlight the car's dynamism and hint at driving pleasure. The warm and carefully finished interior takes up to five people in comfort and, in precise response to customer demand, the new model can be ordered in a choice six different interior trim levels and three equipment levels.

Article Text: Extraordinary room and comfort

Article Text: With a length of 3.99m, Clio III is not only bigger but it also ensures exceptional

Aktueller Absatz: Gesamt:

X, Y, Z
Name: Text167
Geometrie
X–Position: 9.903 mm
Y–Position: 180.976 mm
Breite: 190.000 mm
Höhe: 102.067 mm
Drehung: 0.00
Ursprung:
Anordnung 163
Form
Text
Bild
Linien
Farben

Figure 2.2: Open Source Desktop publishing with Scribus

ing bibliographies, but miss support for advanced layouts and formulas. Others are perfect for graphical tasks like flyer and poster editing, but require all text to be written into special text boxes. The ability to edit documents in a WYSIWYG way is reached by focusing on a certain area.

2.1.5 Typesetting and Wikis

Wikis do not really fit in here. They lack support for typesetting at all. But they can be used to create and edit documents online using a web-based interface, and this is a feature missing in other software.

Other features also useful for document writing are:

- Collaborative writing
- Page history including version differencess
- Restoring old versions
- Information about recent changes in the Wiki

- Web-based access, useable for intranet and internet
- Document splitting and interlinking
- Simple markup instead of WYSIWYG
- Plugin support

Not all Wikis support have the same features, but most of them support basic text formatting like headings, lists and paragraphs. Other frequently supported features are images and tables. Some Wikis like JSPWiki even allow inserting footnotes and references to other sections. What is missing, are layout options like multi-column text. Wikis focus primarily on content.

Unlike word processors and DTP software, Wikis are markup-based. Text formatting is done using markup code similar to HTML, but shorter and simpler. This enables non-technical users to edit Wiki content after a very short learning period. Wiki markup can also be compared to LaTeX code, though LaTeX is much more powerful.

2.1.6 Conclusion

There exist many different tools that are more or less suitable for writing text-based documents. Plain typesetting systems like TeX offer the best output quality, but are markup-based and hard to learn. Desktop publishing tools offer similar quality but are often better for editing graphical documents like flyers. Word processors are wide-spread and easy to use, but the typographic quality cannot reach that of TeX. There is no best tool for all jobs, but it depends on the type of the document.

Wikis are not even typesetting systems, but offer many useful features like collaborative writing. The Wiki way of document authoring can not be reached using any of the other software types. Adding typesetting functionality to Wikis would provide us with a new, powerful tool for document creation. The best typographic quality, on the other hand, is offered by TeX. Adding support for TeX export in JSPWiki would mean to combine the best of two worlds.

The most important requirements are fulfilled:

- Both Wiki content and TeX are markup based, so conversion from one to another should be possible
- JSPWiki can be easily extended by creating plugins
- TeX is open-source and freely available for all platforms JSPWiki runs on

By translating Wiki markup to TeX code, it should be possible to create output in PostScript or PDF format by simply calling the TeX program. The process could be further simplified by using LaTeX instead of plain TeX.

2.2 JSPWiki export

The next step is to find out whether there are existing solutions to export JSPWiki content to PDF. Building upon existing code can greatly speed up the development process.

2.2.1 PDF Plugin

The first tool found was the PDF Plugin[6]. The name sounds like this tool might be useful. But after a short research, it became clear that this is not what we are looking for. The reasons are:

1. It is not even a real plugin but requires modifying the HTML templates.
2. It requires lots of additional software to be installed
3. It is missing important features. For example tables are not supported correctly.
4. It is based on Apache FOP[7] and uses an approach of converting HTML to XML and using CSS to create PDF layout

The tool follows a totally different approach. The main idea is to convert HTML files to PDF by using CSS to define the PDF layout. This is an interesting idea, but has several drawbacks:

1. Formatting Objects[8] tools like Apache FOP don't know anything about typesetting. They cannot help in choosing a correct font size or line spacing. Instead they require a detailed description of the PDF layout
2. Defining the layout in CSS might simplify the process. But the author of the tool admits ”that there is not a 1:1 translation between CSS and PDF”. CSS was made for HTML, not for printed documents
3. It makes use of the HTML code of the page, not the original Wiki code. Some information is lost

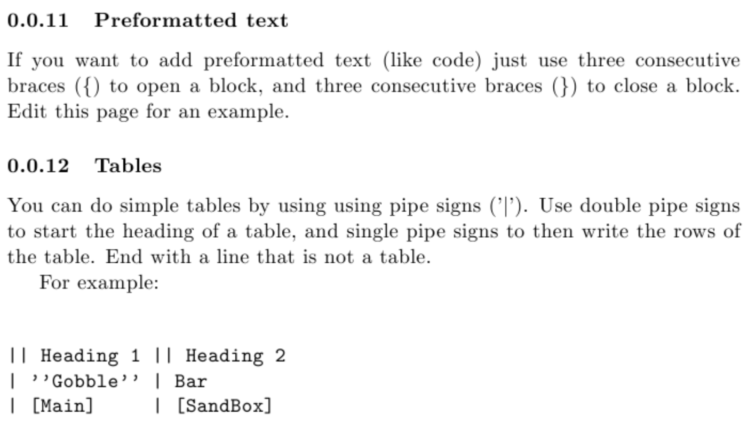
[6] PDF Plugin, http://jspwiki.org/wiki/PDFPlugin
[7] Apache FOP, http://xmlgraphics.apache.org/fop/
[8] XSL Formatting Objects, http://www.w3.org/TR/2001/REC-xsl-20011015/

These points show that this approach is not very useful for our plugin. Using CSS - which was made for HTML - to define a PDF layout and doing the conversion using an Formatting Objects Processor that does not know anything about typesetting, cannot lead to the required high quality output. Additionally, this tool makes use of the HTML code of a Wiki page, not the original Wiki code. So lots of information is already lost. For example there is no way to specially handle other plugins. LaTeX on the other hand was made especially for typesetting, and converting the original Wiki code to LaTeX code should lead to better results.

2.2.2 LatexConverter

Another promising tool was the LatexConverter by Torsten Hildebrandt[9]. The given sample PDF file looked promising. However, this tool is far away from a real PDF export. It is not integrated in JSPWiki, but a simple commandline application. What made the tool interesting were following points:

1. It uses Latex for PDF creation
2. It already converts most of the Wiki markup
3. It is based on JavaCC[10], a parser generator for Java Code



0.0.11 Preformatted text

If you want to add preformatted text (like code) just use three consecutive braces ({) to open a block, and three consecutive braces (}) to close a block. Edit this page for an example.

0.0.12 Tables

You can do simple tables by using using pipe signs ('|'). Use double pipe signs to start the heading of a table, and single pipe signs to then write the rows of the table. End with a line that is not a table.

For example:

```
|| Heading 1 || Heading 2
| ''Gobble'' | Bar
| [Main]     | [SandBox]
```

Figure 2.3: Sample PDF generated by the LatexConverter

Especially the last point is important. Parsing Wiki code can get complex, and using a parser generator should simplify the development of a Wiki parser. By building upon this tool, more

[9] LatexConverter, http://jspwiki.org/wiki/LatexConverter
[10] JavaCC, https://javacc.dev.java.net

development effort could be spent on reliability and features, than on the markup parsing. Integration into JSPWiki should be possible by ignoring the Main class and calling the necessary conversion methods directly.

No other projects of this kind could be found, so the decision was to build the Plugin based upon Torsten Hildebrandts existing LatexConverter.

3 Technologies

3.1 LaTeX

This chapter describes some technical details of LaTeX, especially what has to be considered when creating the LaTeX code or calling the converter program.

3.1.1 A sample LaTeX document

This is how a typical short LaTeX document looks like:

```
\documentclass{article}
\author{Authors name here}
\title{Sample document}
\begin{document}

\section{First section}

\subsection{First subsection}

Some text here...

\end{document}
```

The Wiki code used to build such a document could be:

```
!!First section

!First subsection

Some text here...
```

It is obvious that LaTeX requires additional information not available in the Wiki code, like document type and author. This has to be provided by the plugin. But as LaTeX offers much more features than Wiki markup does, there should be no problem to find equivalent LaTeX code for any Wiki feature. Images, tables, hyperlinks and the other features of JSPWiki are also possible in LaTeX.

3.1.2 Conversion to PDF

After writing the LaTeX documents, it has to be converted to another format. Directly or indirectly (using an intermediate format), LaTeX can be converted to DVI (Device Independent file), PDF (Portable Document Format), Postscript, PNG and others. In our case, the PDF format is chosen.

There are multiple ways of creating PDF documents from a LaTeX source. One option is the *pdflatex* command. A drawback of this method is, that EPS images are not supported. EPS files are often used in LaTeX documents, as they provide high quality images. The *latex* command on the other hand supports EPS but not JPEG and PNG.

In this case, the first method has more advantages. JPEG and PNG images are more important in the web. High quality can also be reached by using high resolution JPEG and PNG images. More information about the different ways of creating PDF documents from LaTeX can be found in [12]. The author also concludes that using *pdflatex* is the best method in most cases.

A problem are GIF images. They are not supported by the common LaTeX programs. Unless they are automatically converted by the LaTeX converter plugin, Wiki users must live without GIF images.

3.1.3 Image quality

Most images that look good on a computer screen, seem to be of bad quality when they are printed. The reason is a low dpi[1] value. To increase the printing quality it is necessary to use images that are large in pixel dimensions and have a high dpi value. How the dpi value effects the printing size and quality is explained in an article[**?**]. The best way to create high quality images is to use a graphic program that supports vector graphics.

[1] Dots per inch

3.1.4 Calling the Converter from Java

The LaTeX programs like *pdflatex* are commandline applications, so they can be called from other programs easily. Java supports calling any system command by using the *Runtime* class. Two things have to be considered:

- Output of the LaTeX program should be fetched to ease error finding
- It has to be ensured that the LaTeX program finishes, otherwise the Servlet waits and cannot even display an error

The first point is not a problem, Java supports fetching the output of system commands. The *pdflatex* error output can also be set to contain file and line number for each error by specifying the *-file-line-error-style* parameter. The second point causes more problems. The *pdflatex* command stops execution and waits for user input when an error occurs. This would mean that the Servlet stops and waits for the *pdflatex* program to complete. The solution is to use *-interaction=nonstopmode* as a parameter for *pdflatex*. This ensures that the LaTeX command does not wait for user input but finishes execution even in case of errors.

3.1.5 Common LaTeX packages

These are LaTeX packages which are commonly used. The LaTeX template files of our converter also makes use of them.

- **KomaScript**: The Komascript package is one of the most well known additional LaTeX packages. It adds better versions of document styles for books, reports, articles and more. They are named *scrartcl, scrreprt* and so on.
- **Hyperref**: The Hyperref package allows additional support for clickable references in the text. It also allows customizing the PDF output. Using hyperref it is possible to set the PDF author and title metadata and to decide whether PDF bookmarks are created.

3.1.6 BibTeX - Using bibliographies in LaTeX

There are two ways to define bibliographies in LaTeX. The first one is by inserting \bibitem commands into the LaTeX document, which is useful for short bibliographies. The other one is by using BibTeX. BibTeX allows it to maintain the whole bibliography in a **.bib* file. The only thing that has to be done, is to create **.bib* file and insert a link to the bibliography in the LaTeX file.

3.2 JavaCC

3.2.1 What is JavaCC?

JavaCC[2] is an open-source parser generator for Java code. A parser generator is a program that reads a grammar specification and creates a parser that accepts input according to the grammar. JavaCC uses BNF[3] grammar as input and outputs Java code. Additionally JavaCC includes JJTree, a tool for building parse trees. JJTree uses a similar grammar to JavaCC.

3.2.2 Parsing using the visitor pattern

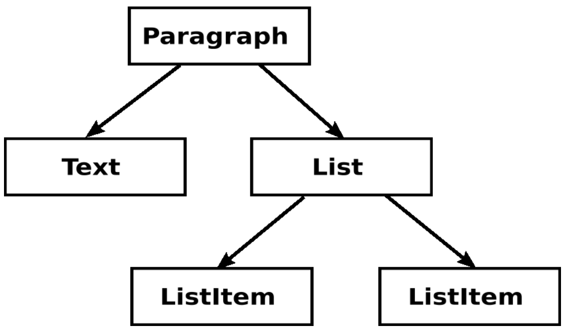


Figure 3.1: An example parse tree

When parsing hierarchical data, the visitor pattern[3] can be used. The idea is to create a tree of nodes representing the data. A Wiki page contains many paragraphs. Each paragraph can contain text, bulleted lists, images, and so on. Thus, the page is represented by a page node containing paragraph nodes. The whole page is seen as a tree.

Visitors are classes that do something with the data stored in the nodes of the tree. This is called *visiting* a node. The visitor travels along the nodes and does something with the data. A visitor can also add, delete or modify nodes in the tree.

For each task, a new visitor can be created. To just output the parse tree to a file, a simple visitor can be created that starts with the root node, writes the name of the node to a file, and continues with the child notes. Other visitors can be created to remove redundant nodes, or to create output in another format - LaTeX in this case.

[2] Java Compiler Compiler, see http://javacc.dev.java.net/
[3] Backus-Naur-Form

3.2.3 Using JJTree and JavaCC

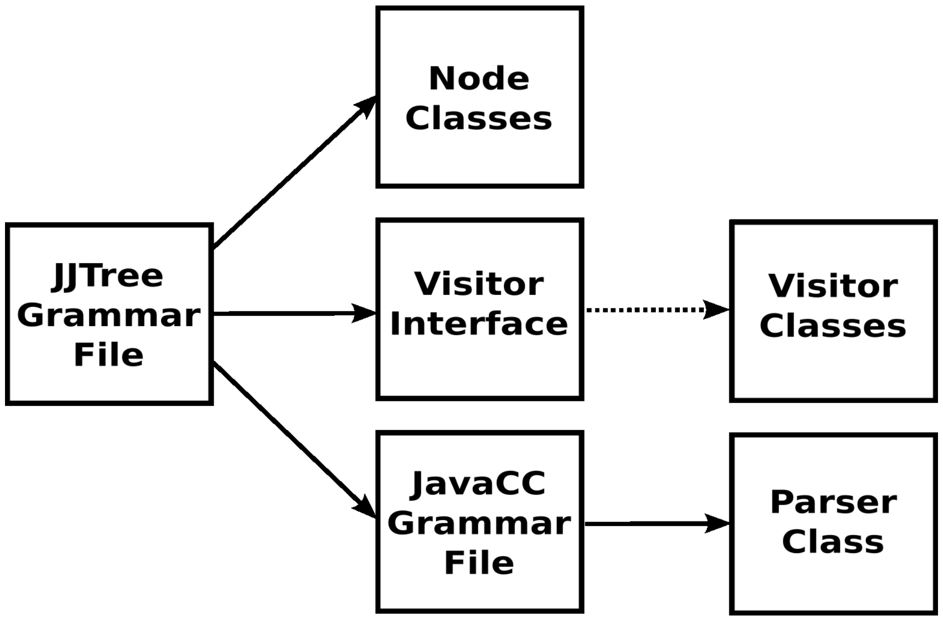


Figure 3.2: The JavaCC development process

JavaCC grammar files are used to create a parser. When using parse trees, a JJTree grammar file is used instead, while the JavaCC grammar is created automatically by the JJTree tool. JJTree could be described as a preprocessor for JavaCC. Besides the JavaCC grammar, JJTree creates the required node classes and the visitor interface. What has to be written by hand are the visitors. They have to implement the visitor interface and contain the operations which are applied to the data.

A sample JJTree grammar file:

```
PARSER_BEGIN(SampleParser)
class SampleParser {

  public static void main(String[] args) {
     SampleParser parser = new SampleParser(System.in);
     parser.ListItem();
  }

}
PARSER_END(SampleParser)

SKIP : { " " | "\n" | "\r" | "\r\n" }
```

```
TOKEN: { < LIST_START : ["#","*"]+ > }

void ListItem() :
{}
{
  <LIST_START> Text()
}
```

This is an incomplete sample grammar file. The parser generated using this grammar can only recognize single list items. What can be seen here, is that the grammar file consists of two parts: The first part consists of java template code definining how the parser works in general. The parser could be a commandline application, or a library class that is called from other programs. The second part is the actual grammar definition specifying the structure of the data that should be recognized.

This sample parser only recognizes single list items. But by adding more definitions for headings, paragraphs, images and the other Wiki features, a complete Wiki parser can be created.

4 Concept

4.1 Overview

Two tasks have to be solved by the LaTeX solution: The first one is the actual conversion of Wiki markup to LaTeX and PDF. The second is to provide a solution for bibliography management within JSPWiki.

4.1.1 Converting the Wiki markup

The most important part is the LaTeX converter. Usage should be as simple as: Inserting a plugin somewhere on a page, defining which pages should be converted, and then pressing a button to create the PDF document.

Requirements for the converter in detail:

1. It has to be a normal JSPWiki plugin, to make it easy to add it to any page
2. Created PDF documents should be automatically attached to the Wiki page containing the plugin
3. It must be possible to define which pages are combined to the result document
4. Complete Wiki markup and bibliographies should be supported
5. There should be a way to choose between different output formats, like article, report, and book.
6. Experts should be given a way to customize the output

4.1.2 Managing bibliographies

LaTeX supports bibliographies, but JSPWiki doesn't. To allow the user to manage bibliographies within JSPWiki, a bibliography management solution is required. JSPWiki should display the bibliography on a Wiki page and allow references from all pages to the bibliography entries. References are displayed as hyperlinks. The *Cite* and *Bibliography* add this functionality to JSPWiki. Both plugins are also used by the LaTeX converter to create the equivalent LaTeX code.

Requirements for the bibliography solution in detail:

1. Not a full-featured user interface, but a simple solution
2. Referencing individual bibliography entries must be possible from anywhere in the wiki
3. The bibliography should be easy to parse for the LaTeX converter

The bibliography solution is very simple and will not be described further. The architecture chapter instead concentrates on the description of the conversion process. The plugin documentation however explains how to use the *Bibliography* and *Cite* plugins.

4.2 Architecture

4.2.1 Frontend and backend

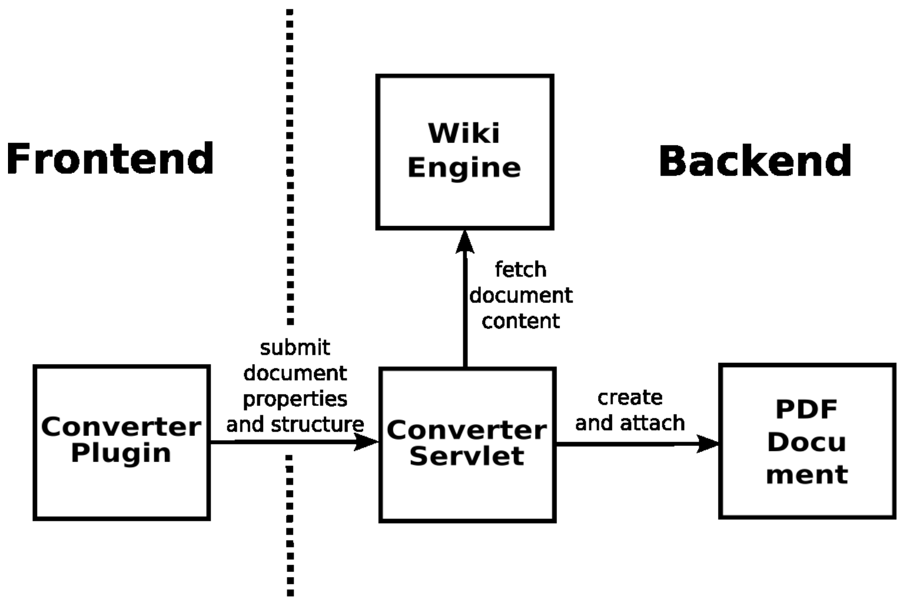


Figure 4.1: The plugin and the Servlet working together

The converter is split into two parts:

- The frontend: A JSPWiki plugin that can be inserted into a Wiki page and adds the possibility to create PDF documents
- The backend: A Java Servlet that does the conversion of the Wiki code to LaTeX and then creates the PDF document

The plugin parameters and the body are used to specify the properties and the structure of the document. When the Wiki page is shown, the plugin displays a HTML formular containing a *Create PDF* button. By clicking the button, the document properties and structure are sent to the Java Servlet. The Servlet uses the *WikiEngine* class of JSPWiki to fetch the content of the requested pages and combines it to a single Wiki document. The Wiki code is converted to PDF and finally attached to the Wiki page containing the plugin.

4.2.2 Integrating the technologies

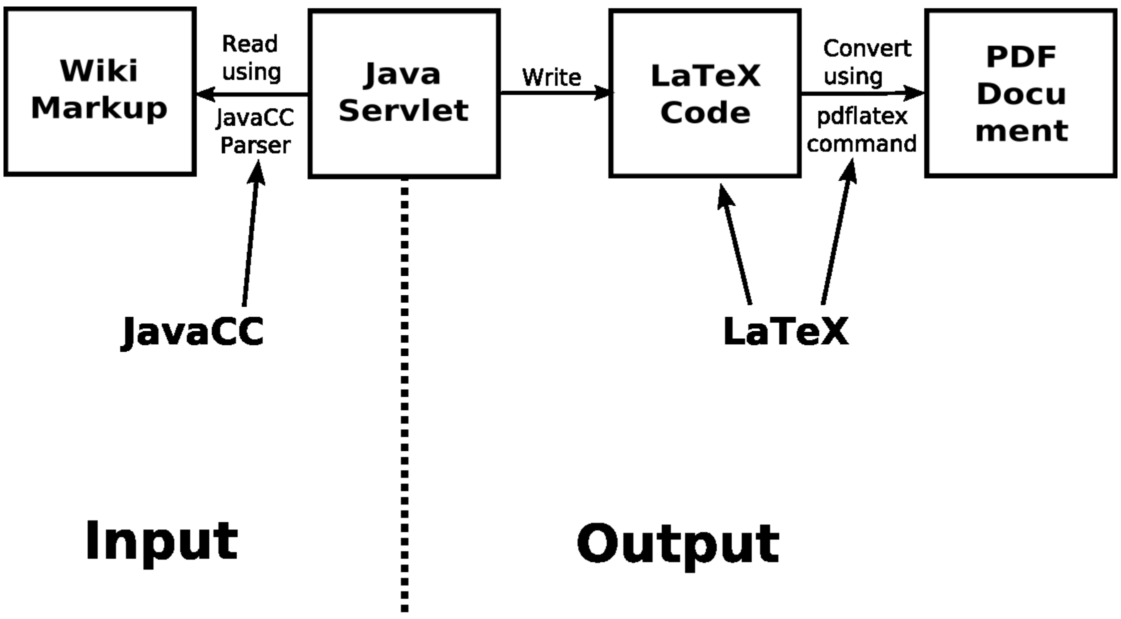


Figure 4.2: Usage of JavaCC and LaTeX for Input and Output

The conversion on the backend is itself done in two steps, one each for Input and Output. This is where JavaCC and LaTeX come into play. On the input side, the JavaCC-based parser reads the Wiki markup and creates the parse tree. The output is created by travelling through all nodes of the parse tree and producing the required LaTeX and BibTeX code. The *pdflatex* program converts everything to the resulting PDF document.

4.3 Configuration

This chapter describes how the plugin can be configured. Configuration here does not only mean to change some plugin settings. Some settings can indeed be changed by modifying the properties file of the plugin. But configuration also means to completely customize the output of the plugin. This can be done in the plugin body or by creating or modifying the template files.

4.3.1 Defining the document structure

Users have to be able to define which pages are combined to the result document. This is done in the plugin body by a custom syntax. This is an example document structure:

```
Introduction
  [Motivation]
  [Goals]
Concept
  [Overview]
  [Architecture]
Conclusion
  [Conclusion]
```

Word surrounded by brackets define which pages of the Wiki are included. The brackets is used for links in JSPWiki, and the document structure uses the same syntax, as it is similar to linking to a Wiki page. Lines without brackets define chapter headings. A more detailed description can be found in the Plugin documentation.

4.3.2 Template usage

Templates prevent hardcoding of output in the source code and simplify changes. The LaTeX Plugin uses three kinds of templates:

- LaTeX document templates
- LaTeX code templates
- HTML form templates

The LaTeX document templates are almost complete LaTeX documents containing placeholders that are filled in by the LaTeX Plugin. The code templates are short LaTeX code snippets that are used to translate Wiki markup to LaTeX. HTML forms are displayed on the Wiki page. They contain the *Create PDF* button and display some additional infos about the chosen settings. The template files contain the HTML forms and placeholders.

LaTeX document templates

A simple template file looks like this:

```
\documentclass{scrartcl}
\usepackage{ngerman}
\usepackage{graphicx}
\usepackage{hyperref}

\author{@@AUTHOR@@}
\title{@@TITLE@@}

\begin{document}

@@CONTENT@@

\end{document}
```

The template defines the type of the document and includes required packages. The placeholders for author, title and the content are replaced by the real values. There are some more placeholders for a bibliography, appendices and so on. Custom placeholders are also possible.

Some default templates are installed together with the LaTeX Plugin. They are stored in files and can only be changed by someone having access to the computer. To enable users of the Wiki to create their own templates, it is also possible to create templates as normal Wiki pages. When a template is loaded, and not found as a template file, the plugin searches for a Wiki page with the given template name and uses the content of this page as template code. If there is neither a file nor a Wiki page with the given name, an error message is displayed.

LaTeX code templates

The code templates are stored in the properties file. They define the LaTeX code for headings, lists, tables and so on. An excerpt of the properties file looks like this:

```
latex.Heading.1 = \\subsubsection{
latex.Heading.2 = \\subsection{
latex.Heading.3 = \\section{
latex.Heading.End = }\n
latex.List.Unnumbered.Begin = \\begin{itemize}\n
latex.List.Unnumbered.End = \\end{itemize}\n
```

These code snippets are used to create the LaTeX output when travelling through the parse tree. The class creating the output looks up required LaTeX code in the properties file and does not need to be modified when simple changes in the output are made. Both the document templates and the code templates allow administrators to customize the default output without changing the program.

HTML form templates

The HTML form template are HTML files with placeholders There exist two templates, one displaying only the button to create the PDF. The other one displays a table showing information about the selected settings like document author, title and the document structure. The compact template only displaying a button looks like this:

```
<!-- Start of @@PLUGIN_NAME@@ Code -->

<form method="post" action="@@SERVLET@@">
  @@PARAMETERS@@
  <input type="submit" value="Create @@FILENAME@@"/>
</form>

<!-- End of @@PLUGIN_NAME@@ Code -->
```

The LaTeX plugin class loads the chosen template and fills the placeholders with real values. The PARAMETERS placeholder is replaced by a list of hidden input fields containing the information that has to be sent to the servlet. Administrators could modify the HTML template to display custom information or add links to a documentation page.

5 Conclusion

5.1 What has been achieved?

To recall the original goals of this study: A plugin should be developed that allows to

- Convert a Wiki page to a high-quality printable document
- Combine multiple Wiki pages to one result document
- Make use of bibliographies and references

All of these goals have been reached. The high quality output of LaTeX makes the resulting documents look very professional. The document structure can be defined by the user using the plugin body. Templates further allow customizing the output. Bibliographies and references can be defined using the *Bibliography* and *Cite* plugins.

If you are reading the PDF or the printed version of this study, then you are actually looking at a document created using the LaTeX plugin. It was written in JSPWiki and exported by the plugin, without further modifying the document by hand.

5.2 Possible extensions

5.2.1 Adaption for JSPWiki 2.4

The plugin was developed for JSPWiki 2.2.33. Newer versions of JSPWiki simplify the plugin installation by defining the plugin searchpath in the jar file of the plugin, instead of the properties file of JSPWiki. The plugin could be updated to make use of this feature.

5.2.2 Handling of GIF images.

GIF images are currently not handled by the plugin, as the GIF format is not supported by the LaTeX programs. They could be automatically converted to JPEG or PNG.

5.2.3 Floating objects

Images and tables are floating objects. LaTeX moves them around to optimize the layout. While this is a very important feature of LaTeX, it is sometimes necessary to keep an image on a certain page. Currently the plugin does not support changing the floating behaviour of LaTeX. There should be an option to define whether an image may float around, or not. There could also be an option to float it around, but only inside the chapter where it is defined.

Bibliography

[1] Adobe Systems Incorporated. PDF reference. http://partners.adobe.com/public/developer/pdf/index_reference.html, 2006.

[2] Allin Cottrell. Word Processors: Stupid and Inefficient. http://ricardo.ecn.wfu.edu/~cottrell/wp.html.

[3] Erich Gamma et. al. *Design Patterns*. 1995.

[4] Dean Jackson. W3C: Printing and the web. http://www.w3.org/2004/02/Printing.html, Feb. 2004.

[5] Donald E. Knuth. The art of computer programming. http://www-cs-faculty.stanford.edu/~knuth/taocp.html.

[6] Donald E. Knuth. *Computers & Typesetting, Volume A : The TeXbook*. 1995.

[7] Leslie Lamport. *LaTeX: A Document Preparation System*. 1986.

[8] Leslie Lamport. Document production: Visual or logical. http://research.microsoft.com/users/lamport/pubs/document-production.pdf, Feb. 1987.

[9] J. H. Saltzer. Typset and runoff, memorandum editor and type-out commands. http://mit.edu/Saltzer/www/publications/CC-244.html, Nov. 1964.

[10] Sun Microsystems. Java serverpages technology. http://java.sun.com/products/jsp/index.jsp.

[11] World Wide Web Consortium. XHTML 1.0 The Extensible HyperText Markup Language. http://www.w3.org/TR/html/, Jan. 2006.

[12] Andrew T. Young. Latex to pdf. http://mintaka.sdsu.edu/GF/bibliog/latex/LaTeXtoPDF.html, 2005.

A Plugin Documentation

A.1 Class LatexConverter

Full name: `org.wikiwizard.jspwiki.latex.LatexConverter`.

The LatexConverter Plugin allows conversion of WikiPages to LaTeX and the Portable Document Format (PDF). You can specify which WikiPages should be converted. This is done by specifiying the pages in the plugin body. The required syntax is explained below. By default (if no plugin body exists), only the current page is included.

A.1.1 Parameters

- **template -** Specifies the used template. This can be one of the templates found in WEB-INF/templates/latex, without file extension, or the name of a Wiki page which contains the LaTeX code to use. This parameter is optional. The configured default template will be used if no template parameter is found.
- **author -** The name of the author of the generated document.
- **title -** The title of the generated document.
- **filename -** The filename of the PDF. Optional, the page name will be used if omitted.
- **mode -** Either ’normal’ or ’compact’. Compact mode only displays a button, normal also incldues a table showing the chosen document settings. This parameter is optional, the configured default mode is used if omitted.
- **abstract -** The name of a wiki page containing the text for the abstract. Optional.
- **bibliography** - The name of a wiki page containing the bibliography. The bibliography page must contain a Bibliography plugin. Optional.

A.1.2 Custom parameters

The templates contain placeholders like @@AUTHOR@@ that are automatically replaced by the corresponding parameter. While `author` is a standard parameter, this is also possible with custom parameter. Just insert @@MYPARAM@@ into a template and then use `myparam='some value'` to replace it with your value.

A.1.3 Handling of plugins

If a plugin is found on a Wiki page, it is ignored, unless it is one of these specially handled plugins:

- `com.ecyrd.jspwiki.plugin.Image` - The referenced image is included into the result document. Only works for JPEG and PNG. The caption parameter is handled. Given width and height parameters are ignored.
- `org.stringfellow.jspwiki.ImageX` - Same as the Image plugin above.
- `com.ecyrd.jspwiki.plugin.Math` - The Math plugin uses LaTeX code, so it's copied into the result document and surrounded my a Math environment.
- `com.ecyrd.jspwiki.plugin.InsertPage` - Included pages are parsed and included in the result document.
- `org.wikiwizard.jspwiki.latex.Bibliography` - The BibTeX code found in the plugin body is written to a bibtex file and the bibliography is included into the result document.
- `org.wikiwizard.jspwiki.latex.Cite` - The Cite code is converted to a LaTeX bibliography reference.

A.1.4 The document structure

This section describes how to define the document structure. Examples are given below.

The document structure is defined in the plugin body. Including pages is done by creating a link to them. To include a page named *Introduction*, a line `[Introduction]` is inserted in the body. The largest heading found on the *Introduction* page is mapped to a section heading. Smaller headings are subsections, and so on. One page can contain many sections, or no section at all. So it is also possible to split sections across multiple pages and including them all.

To include subsections, a `*` symbol is inserted before the link, like `*[PageWithSubsections]`. That means, the largest heading on the included page is mapped to a subsection. This can be used to split sections in several subsections.

Chapters are not included. Instead just the chapter heading is given and the content of the chapters is included as normal sections, as described above. Typically a chapter heading is followed by one or more links to Wiki pages, which define the sections of the chapters. It depends on the used template, whether chapters are used. When using a book or report template, chapters are necessary. If no chapters are defined, all sections have numbers from *0.1* to *0.n*. On the other hand, when using an article template, no chapters are allowed.

A.1.5 Examples

```
[{LatexConverter author='Steffen' title='Test document'

 [SamplePage]
 [AnotherSamplePage]
}]
```

This code is used to create a PDF of two WikiPages. The default template is used, typically this is an article. No chapters are used, as article only have sections.

```
[{LatexConverter author='Steffen' title='Test document'
  template='MyTwoColumnReport' abstract='AbstractPage'
  mode='compact'

Chapter 1
 [Chapter1Part1]
 [Chapter1Part2]
Chapter 2
 [Chapter2IntroductionSection]
  *[Chapter2Subsection]
  *[Chapter2SecondSubsection]
 [Chapter2AnotherSection]
------
Sample code
  [SampleCode]
}]
```

Here a custom template is loaded from a Wiki page containing LaTeX code. An abstract is included from another wiki page. The document consists of two chapters. The first chapter is splitted in to two pages. Note: That doesn't mean the chapter consists of two sections. Instead, the two pages can contain any number of sections and subsections.

The second chapter is splitted in four pages. It starts with an introduction section. The subsections are defined on their own pages. Then another section follows. Note: The last section can also contain subsections, if they are defined directly in the included section page.

Everything following a line starting with "–" is an appendix and given letters from A to Z instead of numbers.

A.2 Class Bibliography

Full name: `org.wikiwizard.jspwiki.latex.Bibliography`.

The Bibliography plugin, combined with the Cite plugin, allows managing a bibliography by using BibTeX code. The Bibliography plugin reads BibTeX code inserted into the plugin body, converts it to HTML and displays a formatted bibliography on the wiki page. HTML anchors are inserted to allow the Cite plugin to link to specific bibliography entries.

This plugin is converted to LaTeX code when parsed by the LatexConverter plugin.

A.2.1 Parameters

- **none**

A.2.2 Examples

```
[{Bibliography

@book{Some99,
author={Author, Some and One, Another},
title={Using bibliographies in WikiWikis},
publisher={TexPublish},
year={1999}
}
}]
```

This bibliography contains one entry and is displayed as

- Some Author and Another One. *Using bibliographies in WikiWikis*. TexPublish, 1999.

A.3 Class Cite

Full name: `org.wikiwizard.jspwiki.latex.Cite`.

The Cite plugin, combined with the Bibliography plugin, allows managing a bibliography by using BibTeX code. The Cite plugin is used to reference bibliography entries. It generates a HTML link which points to the specific entry in the bibliography.

This plugin is converted to LaTeX code when parsed by the LatexConverter plugin.

A.3.1 Parameters

- **src -** Takes a name of a bibliography entry as specified in the BibTeX code, e.g. `Some99`.
- **page -** The page parameter defines the page where the bibliography is found. This parameter is optional. The current page is used, if no page is defined.

A.3.2 Examples

```
[{Cite src='Some99'}]
```

This code references a bibliography entry on the current page.

```
[{Cite src='Some99' page='Bibliography'}]
```

This code references a bibliography entry on the page named `Bibliography`.

B Installation

B.1 Preface

If you haven't already installed a LaTeX distribution, installing the LaTeX plugin will take some time. A LaTeX distribution typically needs several hundred MB of hard disk space. Installing LaTeX and the needed packages will probably the most difficult step, depending on your knowledge and the chosen operating system and LaTeX distribution.

Of course installing LaTeX has to be done only one time, to use it with several JSPWiki installations. Also as all the work is done on the server, nothing has to be installed on client computers, to use the plugin.

Note: The plugin has been developed and tested under Gentoo Linux using the teTeX distribution, version 2.0.2 and JSPWiki 2.2.33.

B.2 Prerequisites

- JSPWiki
- A LaTeX distribution, including the commands
  - pdflatex
  - bibtex

Following LaTeX packages are used:

- Komascript (scrartcl, scrreprt, scrbook)
- ngerman
- graphixc

- hyperref
- times
- eurosym
- ucs [1]

B.3 Installing the plugin

Assuming you have installed all the needed prerequisites, the steps to install the plugin are simple:

1. Unzip the LaTeX Plugin zipfile (`latex.zip`) into the the `JSPWIKI/WEB-INF` directory
2. Configure `JSPWIKI/WEB-INF/latex.properties` to your needs
3. Edit `JSPWIKI/WEB-INF/jspwiki.properties` and add `org.wikiwizard.jspwiki.latex` to the plugin search path
4. Edit `JSPWIKI/WEB-INF/web.xml` and include the LaTeX Servlet by adding following code right after the other servlet declarations

```
 <servlet>
   <servlet-name>LatexServlet</servlet-name>
     <servlet-class>
       org.wikiwizard.jspwiki.latex.LatexServlet
     </servlet-class>
</servlet>

 <servlet-mapping>
    <servlet-name>LatexServlet</servlet-name>
    <url-pattern>/LatexServlet</url-pattern>
</servlet-mapping>
```

[1] You will probably have to download and install it yourself, see http://www.unruh.de/DniQ/latex/unicode/