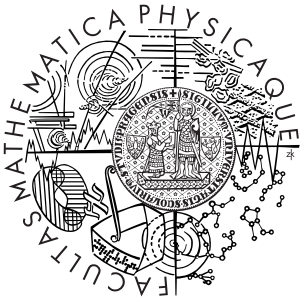
Charles University, Prague, Czech Republic
Faculty of Mathematics and Physics

BACHELOR THESIS



Michal Kebrt

Word-to-LaTeX convertor

Department of Software Engineering
Advisor: RNDr. Tomáš Skopal, Ph.D.
Program in Computer Science

2006

I hereby certify that I wrote the thesis myself, using only the referenced sources. I agree with lending the thesis.

Prague, May 20, 2006 Michal Kebrt

Contents

Název práce: Konvertor Word-to-LaTeX
Autor: Michal Kebrt
Katedra: Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Tomáš Skopal, Ph.D.
E-mail vedoucího: tomas@skopal.net

Abstrakt: V předložené práci popisuji program *Word-to-LaTeX* – konvertor převádějící dokumenty ve formátu *Microsoft Word* do formátu LaTeX, který je vhodný pro sazbu knih, skript, vědeckých článků, atp. Program je však konfigurovatelný do té míry, že umožňuje převádět dokumenty i do zcela odlišných formátů (např. XML). Součástí práce je srovnání textových procesorů a formátu LaTeX, vyzdvižení jejich výhod a nevýhod. Stručně jsou popsány základy objektového modelu programu *Microsoft Word*, možnosti jeho použití, několik jeho problémů a omezení a způsob jak urychlit aplikace, které jej využívají.

Klíčová slova: LaTeX, Word, XML, konverze

Title: Word-to-LaTeX convertor
Author: Michal Kebrt
Department: Department of Software Engineering
Supervisor: RNDr. Tomáš Skopal, Ph.D.
Supervisor's e-mail address: tomas@skopal.net

Abstract: This work is devoted to *Word-to-LaTeX* program that converts documents written in *Microsoft Word* into LaTeX format which is suitable for typesetting books, manuscripts, scientific articles, etc. The program can be customized so much that it enables to produce completely different output formats (e.g. XML). In this work I also tried to compare text processors and LaTeX format and emphasise their pros and cons. The *Microsoft Word* object model is briefly described, its problems and limitations are also covered. Finally, the way of improving performance of applications that automate *Word* is suggested.

Keywords: LaTeX, Word, XML, conversion

Preface

Word-to-LaTeX is a program that converts *Microsoft Word* documents into LaTeX format which is suitable for typesetting books, manuscripts and other kinds of documents, or contributing papers to a lot of conferences.

Although the conversion to LaTeX was the only goal of the project, I tried to make the program as much customizable as possible which resulted in the convertor that supports two output format families – LaTeX and XML. Other markup formats can be easily added through the configuration.

The program is divided into a couple of components which allowed to create a separate command-line convertor, a graphic user interface that's running the command-line convertor, and also a COM object that enables to use the convertor directly from the *Word* application.

The work has two main parts, the first one contains three important chapters. Chapter 1 compares text processors and LaTeX as two different approaches to making documents. It also summarizes *Word-to-LaTeX* features and the possibilities of conversion between *Word* documents and LaTeX format. Chapter 2 describes the implementation. It covers the concept of the convertor and its components, the most important algorithms, and the way of communication between the convertor and the *Word* application. A short overview of the *Word* object model, its problems and limitations is also included. *Word-to-LaTeX* program is compared with all existing *Word* to LaTeX convertors in Chapter 3.

The second part is user related, it comprises of the user's manual and appendices that show sample documents converted with *Word-to-LaTeX* and describe the structure of configuration files.

Part I

Chapter 1

Word to LaTeX conversion

1.1 Word versus LaTeX

At the beginning it wouldn't make sense to compare two particular software products as examples of two very different approaches to making documents. At first it's important to realize how documents are usually created and how they should be created correctly.

Alan Cottrel, in his flammy article [1], very strictly separates two tasks while creating documents.

> The composition of the text itself. By this I mean the actual choice of words to express one's ideas, and the logical structuring of the text. It includes matters such as the division of the text into paragraphs, sections or chapters, adding of special emphasis to certain portions of the text, and so on.
>
> The typesetting of the document. This refers to matters such as the choice of the font family in which the text is to be printed, and the way in which structural elements will be visually represented. Should section headings be in bold face or small capitals? Should the text be justified or not? And so on.

Apart from the fact that in these days the author and the typesetter is often the same person, the author should always mainly concentrate on the first of these tasks. At the beginning there shouldn't be a reason for the typesetting to be an important job.

These two tasks have been put together in widely used WYSIWIG[1] text processors. *Microsoft Word*, *WordPerfect*, *OpenOffice.org Writer* and many more are examples of these programs. They allow to create documents, their design, and layout interactively selecting from a great variety of commands in the program menu. A user always sees a document in its final form because all the document formatting is displayed on the screen (for example, a heading appears in a bold and bigger font). At first sight this feature looks nice, but the "on the fly typesetting" brings a couple of problems which will be summarized below.

LaTeX [3], on the other hand, is a document preparation system which is used for typesetting science and mathematical documents in a high typographic quality.

[1] what you see is what you get

The system is suitable for creating many different kinds of documents, from plain letters to large books. LaTeX is also a standard for contributing manuscripts to a lot of (scientific) conferences.

LaTeX uses TeX [2] – "typesetting system for creating beautiful books", which was developed by professor Donald E. Knuth. LaTeX is actually only a package of macros that make the work with TeX easier. Other sets of macros, which can be used instead of LaTeX, are $\mathcal{AMS}$-TeX and $\mathcal{AMS}$-LaTeX.

What is the main difference between LaTeX and *Word*? When you make a document in LaTeX, you write all the text and commands directly into a plain text file, and you cannot see the final document until you run a program which generates a PostScript or PDF file. Documents can be structured using a lot of special commands (for example `\section{My example section}` command makes a section). Most of modern text editors highlight LaTeX commands so it's never difficult to write and maintain LaTeX documents.

```
\documentclass[11pt]{article}
\begin{document}
\title{Simple \LaTeX{} Document}
\author{Michal Kebrt}
\date{3rd Apr 2006}
\maketitle
\section{Introduction}
This is a simple document created using \LaTeX.
\end{document}
```

Figure 1.1: Simple LaTeX document

Disadvantages and limitations of text processors:

- When writing a large document like a book, text processors often become very slow and documents hardly maintainable due to the real-time typesetting which requires a great amount of memory.
- Authors usually tend to use various kinds of fonts, emphasis, indentation, alignment of paragraphs, and so on. Of course, they do it with an intention to make documents "nicer", but this inconsistence always causes worse readability of documents.
- Authors often forget to concentrate on the content and logical structuring of documents.
- Documents are usually stored in binary files which sometimes cannot be opened without the text processor installed on your machine. There may be also problems with exchanging files between different versions of the same text processor.

Advantages of text processors:

- They are easy to learn and use for most of people. When you want the portion of text to be in a bold font, you just select it, click on the icon, and see the change.
- Users always see documents in their final form. But sometimes this can be a disadvantage which was mentioned above.
- Most of text processors are capable to structure documents using predefined or user styles. It's a pity that WYSIWYG model makes users not use this effective feature.
- Easy insertion of images and variety of external objects (graphs, drawings, and so on).

LaTeX advantages:

- Users can use predefined document templates (e.g. for articles and books) with a professional look and typographic quality.
- Great facilities for writing mathematical expressions, inserting index and citations.
- It's not necessary to specify documents formatting and look because it depends on the selected document style. Authors write only the commands defining the logical structure of documents (e.g. sections and footnotes).
- Many add-ons (e.g. for inserting graphics or hyperlinks).
- Documents are stored in plain text files which can be opened and edited in any text editor.
- Wide portability of TeX and LaTeX system.
- LaTeX is free.

LaTeX disadvantages and limitations:

- It's very difficult to make complex tables with a lot of merged rows and columns.
- Not WYSIWYG model may be a problem for some users. But there are programs like *LyX* which allow visual editing of LaTeX documents.

1.2 What to expect

It's not possible to perform "1:1 conversion" because *Word* and LaTeX are very different document preparation systems. The most important task is surely to convert all the text content. It especially means to convert special and national characters correctly (e.g. $\rightarrow$, σ).

Conversion programs will produce the better results the better input *Word* documents are structured and formatted. This is the reason why people should

use paragraph styles and appropriate *Word* functions for inserting footnotes, sections, index, etc. Once users follow these rules, conversion programs can properly convert almost every part of a document.

1.3 Internal and external conversion

There are two possible ways how to convert *Word* documents to LaTeX format. A lot of information and also the terminology "internal" and "external conversion" come from the article [4].

Internal conversion is carried out within the *Word* application using its object model. It's not significant whether you use the object model in a VBA macro or in some external program. The most important thing is that all parts of documents and information about documents including formatting, *Word* application settings, etc. is available.

Examples of programs that perform internal conversion are *Word2TeX* [18] and *Word-to-LaTeX*.

External conversion, on the other hand, is performed without the help of the *Word* application and its object model. Then we can use at least two methods to convert a *Word* document into LaTeX – either directly access the *Word* document as a binary file or save the document in a more accessible format (often RTF), and then convert it into LaTeX. External conversion has one big disadvantage in comparison with internal conversion. It's usually impossible to retrieve all information about documents, especially about their logical structure.

The first method is completely independent on the *Word* application, so it can be performed outside the *Windows* environment. Although the idea of parsing a *Word* binary file is rather unimaginable, there are a few programs that use this method: *Antiword* [14] and *wsW2LTX* [13]. *rtf2latex2e* [17] is an example of a program that converts RTF documents into LaTeX. .

1.4 Word-to-LaTeX convertor

Word-to-LaTeX performs so-called internal conversion since it uses *Word* object model to retrieve all parts of documents. The lists of implemented features follow.

1.4.1 Most important features

- The conversion can be run from the command-line, through the graphic interface, or directly from *Word*. The latter way of running is much faster than the previous ones.

- The convertor is not limited only to LaTeX format. The program can be easily customized by changing a configuration file or through the graphic interface. The configuration for XML output is an example of such a customization. Additional XSL stylesheets can be created to have *Word* documents in your own format. Sample XSL stylesheet generating XHTML + MathML + CSS documents has been tested and the output looks very nice.

- Equations inserted through *Equation Editor*, *MathType* and *Word* `EQ` fields are converted. There are a couple of predefined equations output formats (e.g. LaTeX, MathML). Numbered equations are also converted. Optionally, references to numbered equations can be automatically recognized in input documents.
- Both raster and vector images, and even embedded objects like *Excel* graphs are converted to Encapsulated PostScript (EPS) format or to bitmaps (PNG format).

1.4.2 Support for structured documents

- Paragraphs marked as headings using the *Word* built-in styles are properly converted to LaTeX sections (the default mappings can be changed).
- Ordered and unordered lists (even nested), and complex tables with merged rows and columns are converted.
- Footnotes and endnotes are properly converted. Bibliography items can be optionally created from endnotes. They're in fact the only way how users can insert bibliography and citations into *Word* documents.
- The program converts table and figure titles, index, table of contents, multicolumn sections, hyperlinks.
- Bookmarks, references and page references to bookmarks are also converted.

1.4.3 Documents formatting

- Mappings between user styles (both paragraph and character) and LaTeX commands can be defined (e.g. style named "preformated" to `verbatim` environment). A special command for each style can be optionally created to make later changes in documents easier.
- Converts various font styles – bold, italic, small caps, subscript, superscript, uppercased, underlined, strikethrough, and hidden. Text written in basic fonts from sans-serif and courier families is also marked in output documents.
- LaTeX font size cannot be easily set exactly the same as in *Word*, so there is a point range that each LaTeX command covers (e.g. 8 – 10 pt for `\small`). The default ranges and commands can be, of course, changed.
- Colored text, highlighted text, and colored backgrounds of table cells can be converted. Borders (even colored) applied to portions of text are also taken into account.
- Paragraphs are converted even with alignments and indentations.
- Line breaks and page breaks are correctly converted.
- Page size and page margins can be converted.

1.4.4 Miscellaneous options and features

- Special and national characters (e.g. Greek, Russian or Hebrew) are converted, even those from the *Symbol* font.
- Editable document preamble; macros like `@WL-DOC_AUTHOR` used in the preamble are replaced with the respective information from *Word* documents.
- LaTeX commands can be inserted into *Word* documents through `PRIVATE` fields. *Word* ignores them, but they are correctly converted.
- Newline separator can be selected from the following separators – `CRLF`, `CR`, `LF`. Lines in output files can be wrapped after each x characters (x is defined in the configuration).

Chapter 2

Implementation

2.1 Basic overview

Word-to-LaTeX performs so called internal conversion since it uses the *Word* object model [7, 8] to retrieve all parts of documents. Basic information about this model will be given in section 2.1.1.

Microsoft Visual Studio 2003 and `C#` language [5, 6] were chosen as a development environment. The whole project is divided into a couple of subprojects described in section 2.1.2. The program design, interesting algorithms, and limitations of the *Word* object model will be depicted in section 2.2.

2.1.1 Word object model

The object model enables you to control the whole *Word* application and manipulate the documents. Each document can be traversed in a couple of ways and a lot of information can be retrieved using tens of various objects' properties.

You have to add a reference to *Microsoft Word Object Model Library* to be able to use the *Word* object model in your program. Such a program should correctly work with all higher *Word* versions in future, but not with the older versions that don't have all the functionality you may use when developing with a newer object model library.

As you can see in figure 2.1, `Application` and `Document` are the essential objects that every program which automates *Word* needs.

The entire *Word* application is represented by the `Application` object. Although the `Application` object makes a lot of other objects available, only a few of them are so important that you will find them in almost every application that uses the *Word* object model. Figure 2.2 shows three of these essential objects.

Only one document can be active within the *Word* application (`ActiveDocument`). All opened documents are grouped in the `Documents` collection. Each `Document` object (figure 2.3) represents a single *Word* document. It comprises of a couple of collections containing footnotes, endnotes, fields, paragraphs, styles, shapes, and so on.

The `Selection` object represents the currently selected area. This object offers almost the same properties as the `Document` object and a couple of additional properties which are illustrated in figure 2.4. The `Find` property is used very often throughout the whole *Word-to-LaTeX* program. It provides the same

```
class WordSketch {
  static void Main(string[] args) {
      Word.ApplicationClass   wordAppClass;
      Word.Application        wordApp;
      Word.Document           document;
      object                  fileName = @"d:\file.doc";

      object readOnly = false;
      object isVisible = false;
      object saveChanges = false;
      object missing = System.Reflection.Missing.Value;

      wordAppClass = new Word.ApplicationClass();
      wordApp      = wordAppClass.Application;
      document     = wordApp.Documents.Open(ref fileName,
        ref missing, ref readOnly, ref missing, ref missing,
        ref missing, ref missing, ref missing, ref missing,
        ref missing, ref missing, ref isVisible, ref missing,
        ref missing, ref missing);

      // print the content of the first paragraph
      Console.WriteLine(document.Paragraphs.Item(1).Range.Text);
      wordApp.Quit(ref saveChanges, ref missing, ref missing);
}
```

Figure 2.1: Sketch of a program that uses *Word* object model

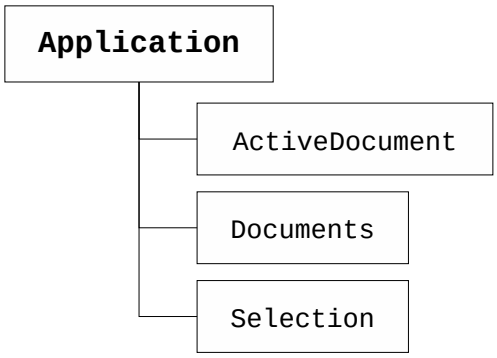


Figure 2.2: Essential properties of the *Word* `Application` object

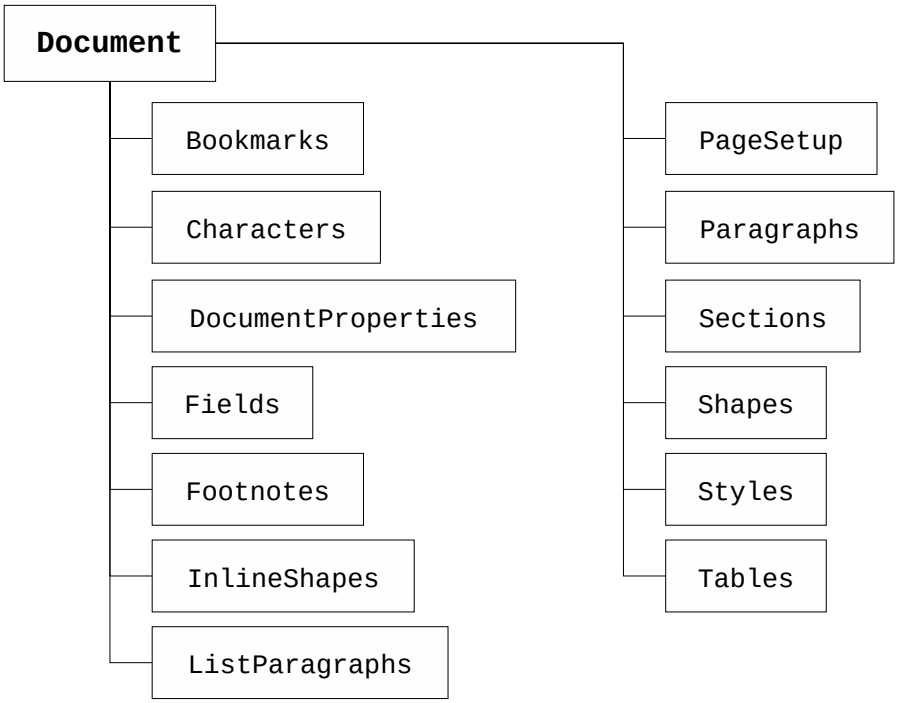


Figure 2.3: Essential properties of the *Word* `Document` object

functionality as the *Word* **Find and Replace** dialog and may help you to find the portions of text written in specified font, color or style, page breaks, tabs, and so on. Even regular expressions can be used when searching for a particular text.

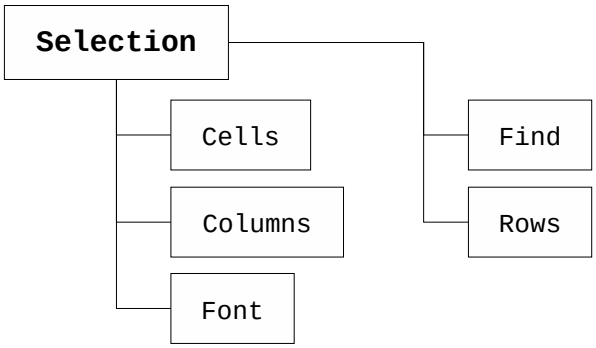


Figure 2.4: Essential properties of the *Word* `Selection` object

The `Range` object is the last one that will mentioned because it's also widely used. This object has nearly the same properties as the `Selection` object. The main differences between the `Range` and `Selection` objects are:

- the `Range` object always represents the contiguous area (it has a start and end position in a document)
- prefer the `Range` to the `Selection` because it's a little bit faster

2.1.2 Components

The whole *Word-to-LaTeX* application is split into 7 projects which allows easy reusing of the source code. Table 2.1 and figure 2.5 show the list of projects.

Project name / output	**Short description**
word-to-latex `word-to-latex-lib.dll`	Library containing all the conversion stuff.
word-to-latex-bin `word-to-latex.exe`	Command-line convertor, uses the *word-to-latex* library.
word-to-latex-configuration-class `word-to-latex-configuration-class.dll`	Library that reads configuration files. It's used in both command-line and GUI programs. Details about configuration files can be found in appendix B.
word-to-latex-glue `word-to-latex-glue.dll`	Simple library containing a class that links the *Word* application with the *word-to-latex* and *word-to-latex-gui* libraries. The class can be used as a COM object directly from a *Word* VBA macro.
word-to-latex-gui `word-to-latex-gui-lib.dll`	Library containing dialogs that enable easy customization of the convertor.
word-to-latex-gui-bin `word-to-latex-gui.exe`	Program that uses the previous GUI library and runs the command-line convertor.
word-to-latex-setup `word-to-latex-setup.msi`	Deployment project.

Table 2.1: List of subprojects

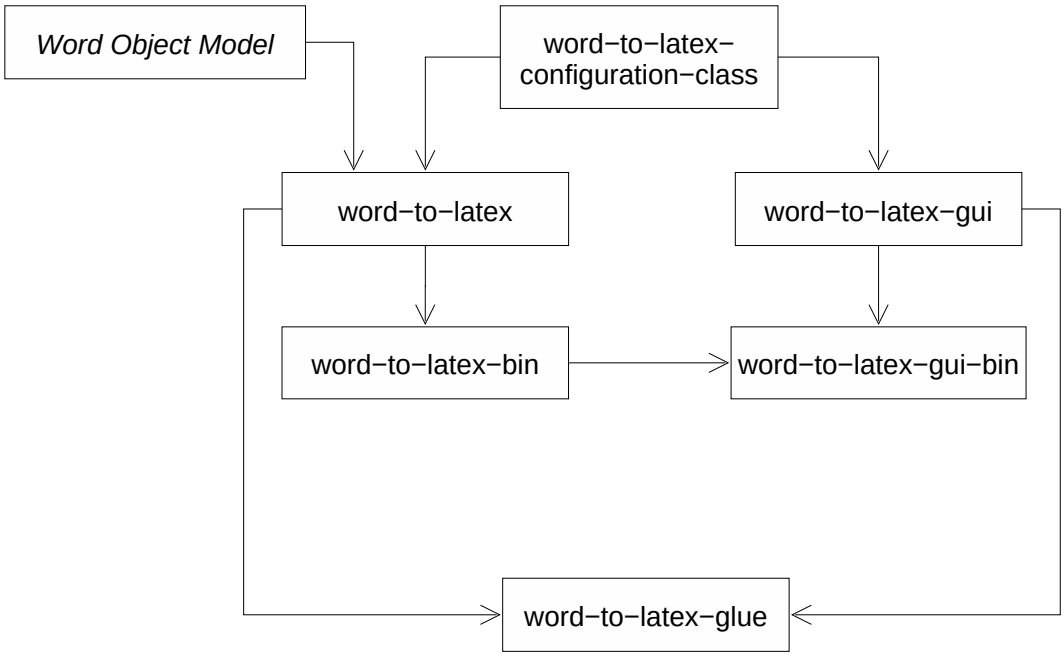


Figure 2.5: Projects dependencies

2.1.3 Libraries

The following libraries are used:

- *Microsoft Word 10.0 Object Model Library*
- .NET *System.XML* for processing XML configuration files and validating them against XML Schema
- .NET *System.Windows.Forms* for creating the graphic user interface
- .NET *System.Drawing* for saving images in PNG format

2.2 Design and algorithms

Projects that worth deeper description are *word-to-latex* which performs all the conversion and *word-to-latex-glue* that allows to run the convertor through a VBA macro.

The `WLConvertor` class, demonstrated in figure 2.6, is the main entry point of the *word-to-latex* library. This class receives an input document, an output filename and a configuration file, initializes the *Word* application and the *MathType* library.

Afterwards, two important tasks are to be done when converting a document. First positions of all special (non-text) elements like footnotes or styles must be retrieved and stored as so-called marks to inner structures of the convertor. Once this is done, the conversion of text content can begin with the document preamble and continue with the document body. Special and national characters are translated to appropriate commands and the marks are inserted to correct positions. More about these two tasks will be told in next sections.

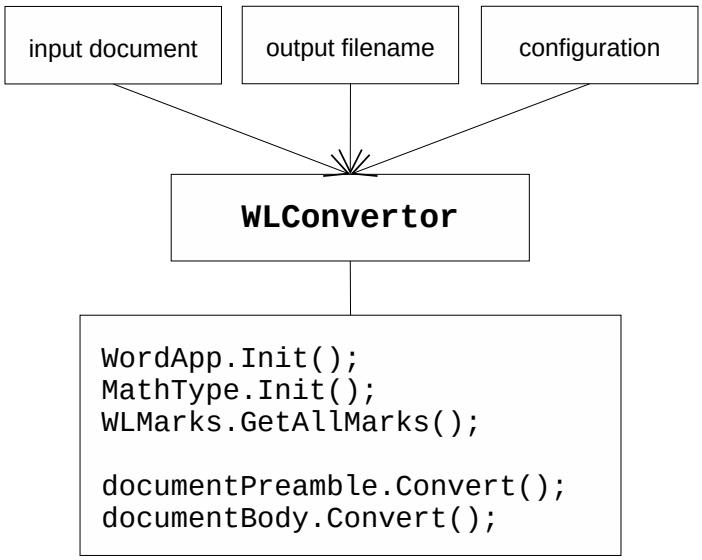


Figure 2.6: `WLConvertor` class

2.2.1 Retrieving and inserting marks

The concept of marks, shortly mentioned in the previous section, is actually the same as so-called XML markup. The convertor retrieves information about a lot of non-text elements contained in the document. Each element like a page break, footnote or text highlight has its start and end position that can be obtained from the `Start` and `End` properties of the corresponding *Word* `Range` object. Like in XML some marks don't need to have end positions. Although this markup concept is very simple, one example in figure 2.7 will make it completely clear.

Lorem ipsum dolor sit amer consectuer.	`<bold>Lorem ipsum</bold>`
	`dolor <linebreak /> sit amet`
	`<font size="3">consectuer</font>.`

Figure 2.7: Markup concept

Each element from *Word* documents has its corresponding class in the *word-to-latex* library. All of them are derived from the `WLDocumentMark` class (figure 2.8). Their instances must have start and end positions, and return commands that will be inserted into these positions. Table 2.2 shows the list of mappings between the document elements, *Word* objects and convertor classes.

Figure 2.8 shows how the marks are collected and stored. The `WLMarks` class contains two queues for the start and end marks (instances of classes from table 2.2). The queue with the start marks is sorted by start positions in the ascending order. The end marks in the second queue are sorted ascending by end positions. Each queue has special rules applied in the situation when the start or (and) end positions of two marks are equal. This prevents from so called "crossover" of the marks (e.g. `<b><i>foo</b></i>`).

Each class like `WLFootnote` has a static member function that loads the marks into these queues. When all the marks are loaded, the convertor can sequen-

	Word object or property		convertor class
footnote	`Footnote`	→	`WLFootnote`
endnote	`Endnote`	→	`WLEndnote`
image	`Shape`, `InlineShape`	→	`WLImage`
bookmark	`Bookmark`	→	`WLBookmark`
TOC	`Field; type=TOC`	→	`WLTOC`
index	`Field; type=Index`	→	`WLIndex`
index entry	`Field; type=IndexEntry`	→	`WLIndexEntry`
hyperlink	`Field; type=Hyperlink`	→	`WLHyperlink`
cross-reference	`Field; type=Ref/PageRef`	→	`WLCrossReference`
equation	`Field; type=Formula/Embed`	→	`WLEquation`
colored text	`Font.Color`	→	`WLColorText`
colored bg.	`Font.Shading`	→	`WLColoredBackground`
style instance	`Range.Style`	→	`WLStyleInstance`
font style	`Font.Bold/Font.Italic/...`	→	`WLFontStyle`
paragraph	`Paragraph`	→	`WLParagraph`
table	`Table`	→	`WLTable`
table cell	`Cell`	→	`WLTableCell`

Table 2.2: Mappings between *Word* objects and *Word-to-LaTeX* classes

tially pick them up from the sorted queues and insert the commands returned by `GetStartCommand()` and `GetEndCommand()` member functions into the output file.

2.2.2 Text content conversion

The conversion of text content follows the marks retrieval task described in the previous section. The `WLDocumentBody` class (figure 2.9) works as a manager – it traverses the document paragraph by paragraph and calls functions for the conversion of tables, list paragraphs, and common paragraphs.

The `WLParagraph` class takes the paragraph text, translates special characters and inserts marks (if any) to appropriate positions in the paragraph. Finally, the converted paragraph can be written to the output file.

2.2.3 Special characters conversion

We must differ between two groups of special characters. The first one contains the characters that have a special meaning in the output format (e.g. "\" in LaTeX). The second group comprises of all national characters and special symbols (e.g. π, $\rightarrow$). The characters from the first group must be always converted earlier because they are often used to translate the ones from the second group.

The way how the characters will be converted completely depends on the configuration described in section B.3. Since *Word* uses Unicode [11] which is also used in configuration files, there is no problem with the conversion of most of characters. Moreover, when the translation is not defined for some character, it can be kept "as is" because the encoding of output files is UTF-8.

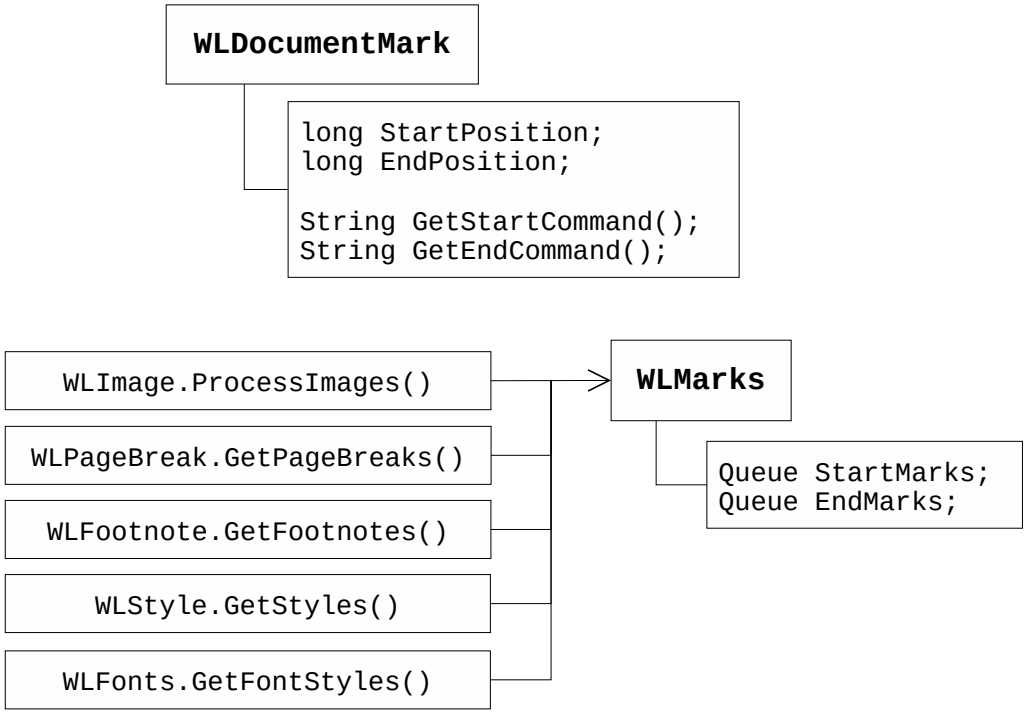


Figure 2.8: `WLMarks` class

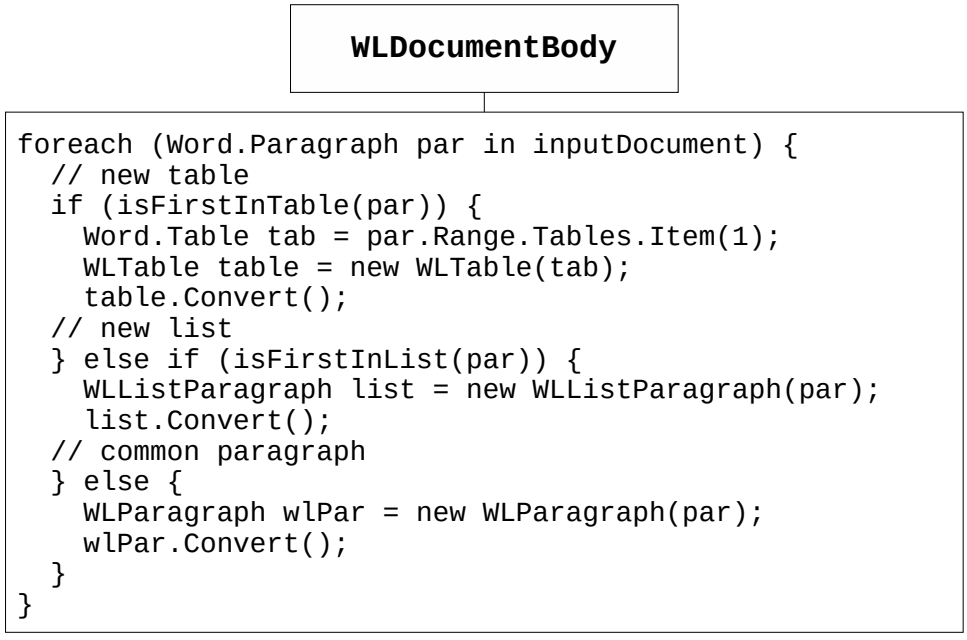


Figure 2.9: `WLDocumentBody` class

Nevertheless the situation is not so clear due to the fonts like *Symbol* or *Wingdings* that have only 0–255 (ASCII) range. Characters from these fonts are internally stored in the part of the Unicode table which is reserved for the application use (`0xF020-0xF0FF`). Currently *Word-to-LaTeX* program has a built-in support for the *Symbol* font. The program defines mappings between most of characters from this font and Unicode. However, it's very difficult to find these characters in documents because *Word* overlaps the real `Font` property with the surrounding font like *Arial* or *Times*. **Find and Replace** and **Insert | Symbol** dialogs have to be invoked to find these symbols and detect their real codes (0-255). Afterwards they can be converted to Unicode following the predefined mappings.

2.2.4 Images conversion

Word-to-LaTeX exports images including embedded objects in two different formats – as bitmaps in PNG format or as vector images in Encapsulated PostScript (EPS) format.

The conversion to EPS format is performed by an external PostScript printer driver (e.g. *Generic Color PS*) which can be easily installed in Windows. The conversion procedure is rather complicated – first the image is copied into the clipboard, then pasted in a temporary *Word* document which is printed to an EPS file using the PostScript printer. Once this is done, the *Bounding Box* property specifying the EPS image size must be edited to match the original image size in the *Word* document. This property is edited without any external program which is quite an easy task. It means to change four numbers in the head of each EPS file (plain ASCII text file).

Example: `%%BoundingBox: 110 687 219 714`

The *Word* object model has no capability to export images. That is why the .NET *System.Drawing* library is used for saving images in PNG format. However, this procedure has one limitation, not all the images can be saved as PNG bitmaps. The *eps2tif* program described in section 5.5 solves this problem.

There's one more way of exporting images as bitmaps. When a *Word* document is saved as a web page, all the images (including embedded objects etc.) are exported as JPEG, PNG and GIF files. As this technique is very laboured, *Word-to-LaTeX* doesn't use it now.

2.2.5 Equations conversion

There are three ways how to insert mathematical expressions into a *Word* document. The first one are `EQ` fields (**Insert | Field**) which can be used even for quite complicated expressions containing sums, brackets, matrices, fractions, etc. `EQ` expressions are written in a source code similar to LaTeX (e.g. `\f(5;3)` makes a fraction). But they have a couple of limitations – for example, you cannot create a triple integral. As there is no API for `EQ` fields, their source code must be parsed to be able to convert them into another format.

Equation Editor (mostly in version 3) is a part of *Microsoft Office* package. It's a visual editor without any mode for writing expressions in a source code similar

to `EQ` fields. In spite of this fact, *Equation Editor* can convert `EQ` expressions into its own format, but not back. The parsing of *Equation Editor* expressions' binary format is the only way of converting them to LaTeX. Although this format is public [9], it's a hard imaginable method for me.

MathType[1] is a professional (and commercial) version of *Equation Editor* with a couple of great improvements – support for numbered equations, automatic recognition of variables, functions and constants, capability to export equations in GIF, EPS, MathML, LaTeX and other formats. *MathType* has an API for basic work with expressions and as it can handle *Equation Editor* and `EQ` expressions too, it's a solution for converting all the expressions within *Word* documents to LaTeX.

Finally we decided to use the *MathType* API for the conversion of equations although it has one big disadvantage – *Word-to-LaTeX* users must have a legal version of this product if they want to convert equations to LaTeX. The possibility of parsing the expressions' binary format was eliminated from our consideration because it would have been a very troublesome task and moreover, the format of *Equation Editor* and *MathType* equations even differs a bit.

`WordToLatex.MathType` namespace contains a few functions wrapping the *MathType* API. *MathType* uses so-called translator files, written in Translator Definition Language (TDL) [9], to export expressions in other formats. It has a couple of predefined translators enabling conversion to MathML, LaTeX and a few other formats.

Word-to-LaTeX tries to recognize simple math expressions written in italics. The following table shows some of the *Word* regular expressions that are currently used.

Regular expression	**Sample matching string**
`[a-zA-Z]`	i
`[0-9]+`	120
`[a-zA-Z]+\(([a-zA-Z]|[0-9]+)\)`	$\sin(x)$

2.2.6 Some nice features

While *Word-to-LaTeX* is converting a document, the user is being informed about the progress of the conversion. It is done through an object that implements simple `ILog` interface. Therefore the console in the command-line convertor and a text box in the graphic interface can be used for printing the log information.

```
public interface ILog {
  // writes a line to the log.
  void WriteLine(string line);
}
```

It is very easy to add new font styles that will be recognized by the convertor. The function `FindFontStyle(ProcessFunction f)` searches for the specified font style in the input document and calls the given handler that has the same arguments as the following delegate.

[1] http://www.dessci.com

```
// range - range in the document that has the given font style
// style - style (character or paragraph) of this range
private delegate void ProcessFunction(Word.Range range,
                                      Word.Style style);

// example of usage
// set the font style and pass the ProcessFontStyleBold handler
WLConvertor.WordApp.Selection.Find.Font.Bold = -1; // true
FindFontStyle(new ProcessFunction(ProcessFontStyleBold));
```

Although the convertor is highly customizable and has a built-in support for two different output format families (LaTeX and XML), there are only a few places in the source code where the convertor handles these output formats differently. Appendix B describes the configuration in details.

2.3 Problems

The problems and limitations of the *Word* object model and *Word* itself will be described in this section.

Sometimes funny

The *Word* object model sometimes behaves funny in a couple of things. Exceptions are ever and again thrown although there is no reason for *Word* to do it. *Word* sometimes gives you completely bad information about the measures of tables and pages. The most funny thing is to get a different output from a VBA macro and identical C# code.

Citations

Word has no tool for inserting citations (e.g. "[1]", "[Ka78]") into documents. Somebody uses endnotes (**Insert** | **Reference** | **Footnote** | **Endnote**) to insert citations and therefore *Word-to-LaTeX* can properly convert them to the `bibliography` environment. The program may also convert the portions of text that match the citation pattern (`\[[A-Za-z0-9_-]+\]`) to the commands specified in the configuration (e.g. "[1]" to `\cite{bib1}`).

Cross-references

Word-to-LaTeX converts cross-references (inserted through **Insert** | **Reference** | **Cross-reference**) only to bookmarks (inserted through **Insert** | **Bookmark**). Other cross-references (to sections, tables, etc.) use *Word* internal codes (e.g. `PAGEREF _Ref133683482`) and cannot be converted.

Colors

The *Word* object model uses two data types for representing colors. `Word.WdColor` type is a standard RGB representation stored in one `long` number, `Word.WdColorIndex` type contains a couple of internal codes. The mapping between `Word.WdColorIndex` and `Word.WdColor` had to be made to be able to convert all the colors to RGB. Colors are written in HTML notation to output files (e.g "`FF0000`" is a red color).

There is no fast way of searching the portions of text that have color different from black. The document has to be traversed character by character and the convertor checks whether the color of the current character is different from the color of the previous one. This technique is extremely slow, so the conversion of colored text can be disabled in the configuration.

Colored backgrounds of text are converted if they are applied to a style or inserted using the **Highlight** tool. Such highlighted text can be easily found with the *Word* `Find` object.

Titles

The *Word* object model doesn't provide any information about links between tables or figures and their titles. Therefore the convertor must check whether the paragraph before the title contains a table or some kind of figure.

Tables

The conversion of tables is surely the most complicated part of the convertor. Its source code has been rewritten a few times, so it's working quite good now, but the code is a little bit messy. The problem is both in *Word* and LaTeX.

The *Word* object model has a very limited interface for tables with merged cells. There are no functions like `GetMergedColumnsCount()` or `GetMergedRowsCount()`. This important information has to be counted on the basis of cells' widths. Moreover *Word* sometimes gives pointless information about various properties of tables, so its really difficult to convert them properly. LaTeX capabilities for making complex tables with a lot of merged cells are not very nice which brings other complications into the source code.

The work with tables becomes much more complicated when there are nested tables in the document, so they are ignored by *Word-to-LaTeX* now.

2.4 Improving performance using COM

Word-to-LaTeX is not very fast when it's converting a large document. Since users usually run the conversion only a few times before they find the best configuration for the particular document, the speed is not the most important feature.

Before the performance can be improved, we must figure out what causes the conversion procedure to be so slow. It's so-called interprocess communication (IPC) between the convertor process (`word-to-latex.exe`) and the *Word* application (`winword.exe`) that is extremely exhausting due to the intensive utilizing of the *Word* object model.

It's good to follow the rules for writing fast *Word* automation programs (e.g. prefer `Range` objects to the `Selection` object), but the speed improvement is not very high. It would be perfect to have only one process when automating *Word*. We could use VBA, but it's not a suitable language for such a big project like this convertor.

The idea of only one process can be easily implemented using the Component Object Model (COM) described in [10]. It's possible to create a simple class connecting the *Word* application with the convertor library, register it as a COM object and then use it in a *Word* VBA macro. There's only one process then (`winword.exe`) and the conversion is much faster.

The `WordGlue` class (figure 2.10) works as such a connector. It receives a *Word* application instance and a document to convert. Afterwards the customization dialog (`MainForm`) is created and the *Word* `Application` object is passed to the `WLConvertor` class described in section 2.2. The user starts the conversion procedure pushing the button in the dialog.

```
public class WordGlue {
    private Word.Application _app;
    private Word.Document _origDoc;

    public void Startup(object app, object doc) {
       _app = (Word.Application) app;
       _origDoc = (Word.Document) doc;
    }
    public void Shutdown() {
       _app = null;
       _origDoc = null;
    }
    public void Convert() {
       MainForm form = new MainForm(true);
       WLConvertor.WordApp = _app;
       form.FromWord = true;
       form.ShowDialog();
    }
}
```

Figure 2.10: `WordGlue` class

Once we have the connector class (`WordGlue`) registered as a COM object, it can be simply used in a VBA macro which illustrates figure 2.11.

```
Sub WordToLatex()
    Dim app As New WordGlue

    app.Startup Application, ActiveDocument
    app.Convert
    app.Shutdown
    Set app = Nothing
End Sub
```

Figure 2.11: Running the convertor from a VBA macro

Chapter 3

Related projects

3.1 Summary

There are a couple of programs that convert *Word* documents to LaTeX. Only one of them (*Word2TEX*) is so good that it will be described in details and compared with *Word-to-LaTeX* in section 3.2. The other convertors are listed only in a brief, more details can be found in [12].

Word independent convertors

- *wsW2LTX* [13] – based on cross-platform *wv*[1] library that allows to access *Word* binary files. The convertor has no customization options, doesn't convert font sizes, user styles, headings, paragraph aligning, etc.
- *Antiword* [14] – wide portable, converts only to plain text or PostScript. Font styles and sizes, footnotes, lists, tables, etc. are converted. Problems with figures sometimes occur.

Convertors that need Word installed

- *GrindEQ* [15] – works as a *Word* add-in. Cannot be customized, doesn't handle lists, headings, font sizes, paragraph indentation, special characters, graphics, etc.
- A couple of very old convertors (e.g. *Word_TEX*) can be found at CTAN sites [16].

RTF to LaTeX convertors

RTF is a document file format that can be read and exported in most of text processors (including *Microsoft Word*).

- *rtf2latex2e* [17] – produces quite nice LaTeX output. It converts font styles, footnotes, tables, paragraph styles, *Equation Editor* equations and some figures.
- Other RTF to LaTeX convertors can be found at CTAN sites [16], but they cannot usually handle the new version of RTF format.

[1] http://wvware.sourceforge.net/

3.2 Word2TEX versus Word-to-LATEX

Word2TEX [18] is the only *Word* to LaTeX convertor that will be compared with *Word-to-LATEX*. All the other convertors are either very old or don't produce good results. *Word2TEX* is a commercial convertor which requires *Microsoft Word* to be installed. Here are the lists of useful features and advantages of both programs.

Word2TEX unique features and advantages:

- A couple of built-in output formats (LaTeX 2.09, LaTeX 2_ε, $\mathcal{AMS}$-LaTeX). *Word-to-LATEX* can produce even more output formats (e.g. XML) because its configuration is not so tied up with LaTeX.
- It can put figures and tables at the end output files.
- *Word2TEX* users can easily define own mappings between math equations and LaTeX commands through the graphic interface. *Word-to-LATEX* users must edit *MathType* TDL files to customize the conversion of equations.
- Commands for *PDFTEX* and for `\maketitle` can be inserted in special dialogs. *Word-to-LATEX* can do the same in the preamble configuration.
- *Word2TEX* is independent on *MathType* when converting equations.
- It can extract figures in original format (e.g. WMF or BMP)
- The conversion is very fast.

Word-to-LATEX unique features and advantages:

- Users can run the conversion from the command-line, through the graphic interface or directly from *Word*.
- The command-line convertor can be used for batch processing of more documents.
- The configuration is stored in plain text XML files which can be easily edited.
- Output files have UTF-8 encoding which is suitable for easy insertion of national characters.
- Paragraph indentation may be converted.
- Page size and page margins are properly converted.
- It converts equations inserted through `EQ` fields.
- Equations can be exported as images which is suitable for users who don't have *MathType* installed.
- The commands defining user styles can be created in the document preamble.

- Each user style may be converted "as is" with no translation applied on the text content of the style. So it's possible to convert the style to the `verbatim` environment.
- LaTeX commands can be inserted into *Word* documents through `PRIVATE` fields.
- Colored backgrounds of text and text borders are optionally converted when they occur in a user style. Colored backgrounds of table cells may be converted, too.
- Bookmarks and page references are also converted.
- The convertor can automatically recognize citations in input documents. Still the bibliography items have to be added manually.
- The default font size of documents can be set.

The most significant difference between *Word2TeX* and *Word-to-LaTeX* is the overall conversion speed. Figure 3.1 shows the times achieved when converting two documents using *Word2TeX*, *Word-to-LaTeX* command-line program and *Word-to-LaTeX* COM object in a VBA script. The machine used for testing was: Athlon 2200+, 1.8 GHz, 512 MB RAM, *Word XP*.

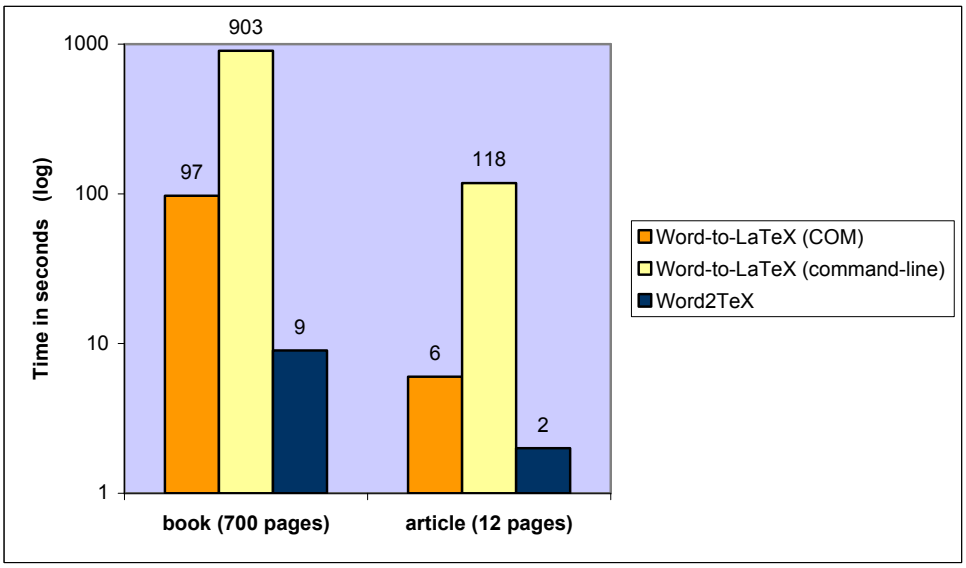


Figure 3.1: *Word-to-LaTeX* vs. *Word2TeX* speed

A couple of problems occurred when I was testing *Word2TeX*.

- Table of contents wasn't converted correctly.
- Some images had wrong width specified.
- Predefined translations for a few Czech national characters were missing.
- A couple of tables and lists were converted badly.

The following pages contain a *Word* document converted with both *Word-to-LaTeX* and *Word2TeX*. Although it's very a short document, a couple of *Word2TeX* limitations are illustrated – badly converted font sizes, courier and sans-serif fonts, no indentations, no background colors, wrong table, page reference is hard-coded (not inserted through `\pageref`).

Word original

Some colors and font sizes

Ied id risus. Donec **enenatis** viverra, velit nisl mattis urna, non luctus sapien ante et leo. Integer **pharetra** congue **tempus metus** sem eu lorem dio vitae nibh. Donec porta

Source code

```
int main (int argc, char * argv[]) {
    if (0 < 1) {
        printf("Hello World");
    }
    return 0;
}
```

Indenting, EQ, page reference

Aliquam egestas, quam in imperdiet imperdiet, nulla nulla lacinia nunc, congue tempus. **EQ** field: $\sqrt{x}$, **PAGEREF**: colors are on page 1.

Table

Blue	Center	Right
ee	*Italics*	Pink

Table 1 Sample table

Converted with *Word-to-LaTeX*

Some colors and font sizes

Ied id risus. Donec **enenatis** viverra, velit nisl mattis urna, non luctus sapien ante et leo. Integer **pharetra** congue **tempus metus** sem eu lorem dio vitae nibh. Donec porta

Source code

```
int main (int argc, char * argv[]) {
    if (0 < 1) {
        printf("Hello World");
    }
    return 0;
}
```

Indenting, EQ, page reference

Aliquam egestas, quam in imperdiet imperdiet, nulla nulla lacinia nunc, congue tempus. EQ field: $\sqrt{x}$, PAGEREF: colors are on page 1.

Table

Blue	Center	Right
ee	*Italics*	Pink

Table 1: Sample table

Some colors and font sizes

Ied id risus. Donec **enenatis** viverra, velit nisl mattis urna, non luctus sapien ante et leo. Integer **pharetra** congue **tempus metus** sem eu lorem dio vitae nibh. Donec porta

Source code

```
int main (int argc, char * argv[]) {
if (0 < 1) {
printf("Hello World");
}
return 0;
}
```

Indenting, EQ, page reference

Aliquam egestas, quam in imperdiet imperdiet, nulla nulla lacinia nunc, congue tempus. **EQ** field: (), **PAGEREF**: colors are on page 1.

Table

Table 1: Sample table

Blue	Center	Right
ee	*Italics*	Pink

Chapter 4

Conclusion

Word-to-LaTeX convertor has almost all the features from the specification[1] that was given in April 2005. A short list of features that were not implemented or could not be implemented due to the *Word* limitations follows.

- Cross-references only to bookmarks are properly converted. The other cross-references use *Word* internal codes and therefore cannot be converted.
- *Word* has no tool for inserting citations, so there is nothing to convert. Nevertheless the convertor may recognize hard-coded citations.
- The character set of output files cannot be changed as was promised in the specification. UTF-8 encoding has been chosen because it covers all national and special characters.
- Images are not exported in original format, PNG and EPS are the only output formats.

A lot of additional improvements have been done, the most important are:

- The convertor is not tied up with LaTeX, so the configuration for XML output could be easily created. Such output files may be transformed to other formats (e.g. XHTML + CSS).
- The convertor can be executed through the COM object. The total performance was increased more than 10 times when this COM object was used in a VBA macro.
- Paragraph and character user styles are translated to appropriate commands.
- Colored and highlighted portions of text can be optionally converted.
- Parts of text written in basic sans-serif and typewriter fonts are properly marked in output files.

[1] http://www.ms.mff.cuni.cz/~kebrm3am/word-to-latex/spec.pdf

Although the convertor has a lot of features, a few other improvements could be done in future:

- The conversion of equations without *MathType* installed. It means to parse the equations' binary format.
- Convert the spaces between paragraphs.
- XSL stylesheets for other output formats (e.g. *tBook* or *Simplified Docbook*) could be created.
- The conversion of nested tables.

Part II

Chapter 5

User's manual

5.1 Requirements and installation

- *Microsoft Windows 2000* or *XP* is required.
- *Microsoft .NET Framework Version 1.1* or higher is required. The installation file can be found in the `setup` directory.
- *Microsoft Word XP (2002)* or higher is required to be installed on your system.
- If you want to export mathematical equations not only as images, but also to LaTeX or MathML formats, you will have to install *Design Science MathType*[1] (it's a commercial product).
- You must have a PostScript printer driver installed on your system to be able to export images to EPS format. You can follow instructions here[2] to add very good *Generic Color PS Printer*.

After you have installed all the required software, close *Word* (if it's running), execute `setup.exe` in the `setup\Word-to-LaTeX` directory, and follow the instructions. You must have administrator privileges to install the whole application properly. Once the installation is finished, you will find a couple of files in your *Word-to-LaTeX* directory. Some of them are listed here:

- `word-to-latex.exe` – *Word-to-LaTeX* command-line convertor
- `word-to-latex-gui.exe` – *Word-to-LaTeX* graphic user interface
- `config.xml, XMLconfig.xml` – convertor configuration for LaTeX and XML output
- `html.xsl` – XSL file which transforms XML output to HTML
- `manual.pdf` – user's manual
- `eps2tif` – directory containing a batch file for converting EPS images to TIF format

[1] http://www.dessci.com/en/products/mathtype/

[2] http://www.princeton.edu/~cavalab/tutorials/computers/postscriptPrinter.html

5.2 Uninstallation

If you want to uninstall *Word-to-LaTeX* from your system, go to **Control Panel | Add or Remove programs** and select *Word-to-LaTeX*. Please close *Word* (if it's running) before uninstalling.

5.3 Configuration

All the program configuration is stored in an XML file with a public format which is defined using XML Schema in the `config.xsd` file. Before the conversion procedure starts, the configuration is validated against the schema, so you must be very careful when editing the file manually.

There are two predefined configuration files in your *Word-to-LaTeX* directory, `config.xml` for conversion to LaTeX and `XMLConfig.xml` for conversion to XML format.

Don't be afraid if XML is an unknown abbreviation for you. There is no need to know anything about XML technologies because you can customize the convertor also through the graphic interface which will be described in section 5.6.

Appendix B describes the XML structure of configuration files and possible values in each element and attribute.

5.4 Command-line convertor

When the command-line convertor (`word-to-latex.exe`) is executed without any parameters, the list of all possible options from table 5.1 will be printed.

```
word-to-latex.exe -i inputFile [-o outputFile] [-opt confFile]
```

`-i`	input file name
`-o`	output file name
`-opt`	configuration file name

Table 5.1: `word-to-latex.exe` options

The only required option is "`-i`". When the output file is omitted, the input file name appended with "`.tex`" extension is taken instead. If the configuration file is not specified, the default configuration stored in the `config.xml` file is used for the conversion.

After you run the program with correct options, it prints all the file names (input, output, configuration) and also your *Microsoft Word* version which can be useful when an error occurs. Then the conversion routine is started and you will be informed about the progress.

Please be patient when you are converting a large document, it can take a long time to convert it. Much more faster way of running the conversion will be described in section 5.7.

5.5 EPS to TIF image conversion

As not all images included in *Word* documents can be converted to bitmaps, I wrote a simple batch file (`eps2tif.bat` in the `eps2tif` directory) which converts EPS files to TIF format. It benefits from the fact that *Word-to-LaTeX* can export all images to EPS format.

This batch file requires *Ghostscript*[3] program which is free for non-commercial use. The path to the *Ghostscript* executable must be specified at the top of the `eps2tif.bat` file.

When you want to export all images from a *Word* document to some bitmap format (PNG, JPEG, and so on), just run *Word-to-LaTeX* to have an EPS version of each image and then execute the `eps2tif.bat` file with the options described in table 5.2. Finally you can convert the output TIF files to the format you prefer (for example *Irfanview*[4] does this very effectively).

`eps2tif.bat inDir outDir`

`inDir`	directory from which the files with `.eps` extension are taken
`outDir`	directory where the `.tif` files will be saved

Table 5.2: `eps2tif.bat` options

5.6 Graphic user interface

For most of users the graphic interface will be the most frequent way of using *Word-to-LaTeX* convertor. To run it, just click the icon on your Desktop or in the Start menu, or execute the `word-to-latex-gui.exe` file in your *Word-to-LaTeX* directory.

After executing the program, the configuration dialog will appear. All the six tabs will be described now.

5.6.1 Running the conversion

Only the **Input document** is required to be selected. When the **Output file** is omitted, the **Input document** file name appended with "`.tex`" extension is taken instead.

Two configuration files can be found in your *Word-to-LaTeX* directory, `config.xml` for conversion to LaTeX and `XMLConfig.xml` for conversion to XML.

When the **Configuration file** is omitted, `config.xml` will be used instead. But be careful, it's recommended to customize the settings for each document you convert. **Save as ...**, **Save** and **Load** commands in the **Configuration** menu can be used to load and save convertor configurations. Remember that the current configuration must be saved before it is applied during the conversion. You can check the option **Save configuration before conversion** to save the configuration automatically after pressing the **Convert** button.

[3] http://www.cs.wisc.edu/ ghost/

[4] http://www.irfanview.com/

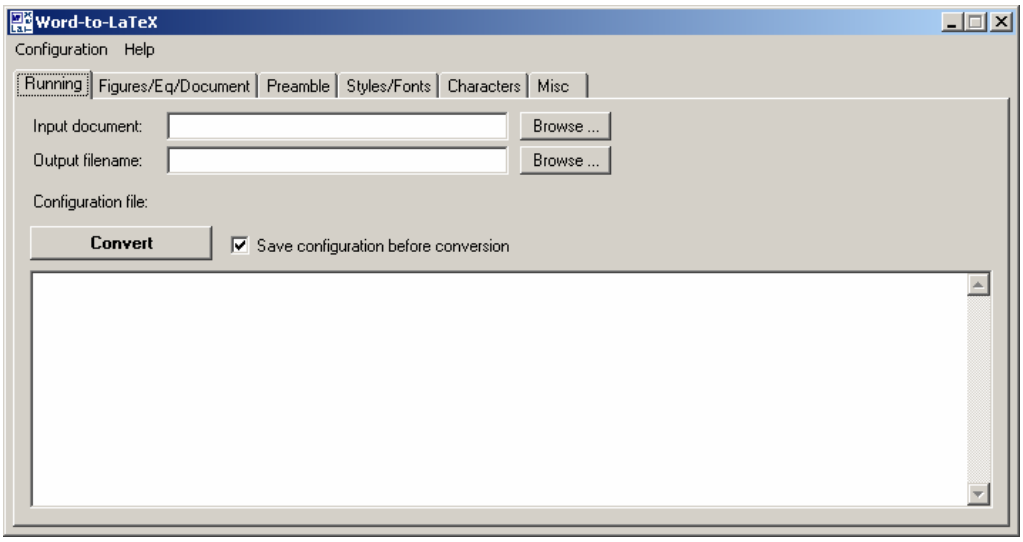


Figure 5.1: "Running" tab

When you press the **Convert** button, all the file names (input, output, configuration) and also your *Microsoft Word* version will be written to the text box below. This can be useful when an error occurs. Then the conversion routine is started and you will be informed about the progress in the text box. Please be patient when you are converting a large document, it can take a long time to convert it. Much more faster way of running the conversion will be described in section 5.7.

5.6.2 Figures, Equations and Translations

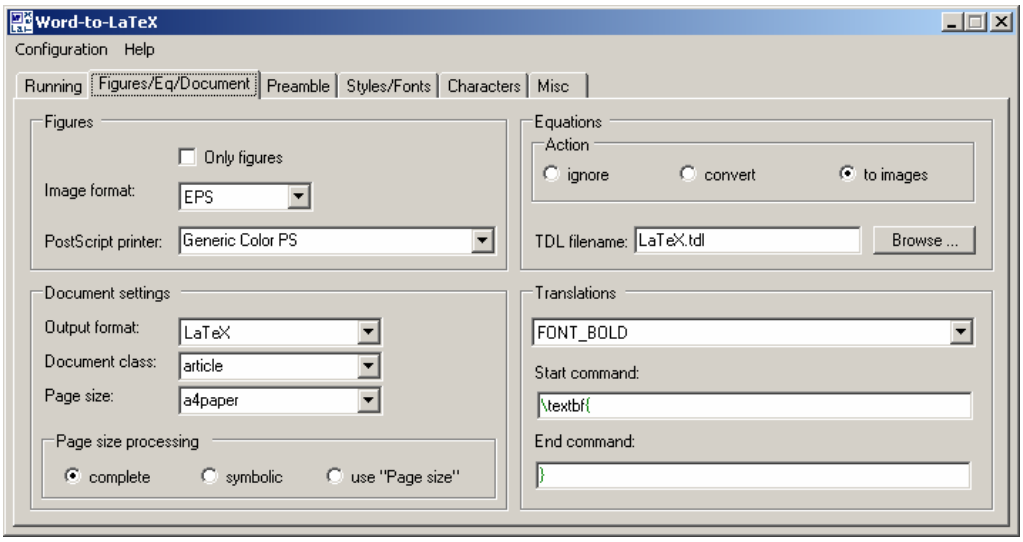


Figure 5.2: "Figures/Eq/Document" tab

Figures

Check **Only figures** to convert only figures and ignore the text content of the input document. *Word-to-LaTeX* exports images (including embedded objects like *Excel* graphs) in two formats – vector Encapsulated PostScript (**EPS**) or bitmap **PNG**. If you want to export images to EPS format, you must specify the **PostScript printer**. This topic was mentioned in section 5.1.

EPS format is recommended because EPS images can be easily integrated into LaTeX documents and moreover some images included in *Word* documents (e.g. *Word* drawings) cannot be exported as bitmaps. If this occurs, the convertor will give you a notice and after it finishes, you can export all images to EPS format and use `eps2tif` program described in section 5.5 to have a bitmap version of each image.

Equations

If you have *MathType* installed on your system, you can check **convert** and all equations inserted through *Equation Editor*, *MathType* and *Word* `EQ` fields will be converted. Otherwise you have to select **ignore** to ignore all equations or **to images** for exporting equations to images.

When the **convert** option is selected, the output format of converted equations depends on the translation file defined in the **TDL filename** box. See the `Translators` subdirectory of your *MathType* directory for possible values. You can edit or add new files to this directory if you want to customize the conversion of equations.

Document settings

As the convertor performs a few special actions depending on the **Output format**, you must select **LaTeX** or **XML**. But remember that it doesn't change any **Translations**.

The `@WL-DOC_CLASS` macro used in the document preamble will be replaced with the value of the **Document class** option. The `@WL-PAGE_SIZE` macro will be replaced with a value depending on the **Page size processing** option as shows table 5.3.

Option name	`@WL-PAGE_SIZE` **will be replaced with**
complete	the complete definition of the page size matching the page size of the input document
symbolic	the convertor will try to translate the symbolic page size (e.g. A4) of the input document to an appropriate LaTeX size (e.g. `letterpaper`)
use "Page size"	the value of the **Page size** option

Table 5.3: **Page size processing** options

Translations

The translation mappings between input document elements and LaTeX commands are defined here. It comprises of headings, font styles, footnotes, tables,

alignments, colors, and so on. Each element has a **Start command** which is inserted before the element itself and an **End command** inserted after the element.

One example: Let "*some text*" appear in the document and the `FONT_ITALIC` mapping is "`\textit{`" for the start command and "`}`" for the end command. Then "`\textit{Some text}`" will be written to the output file.

The complete overview of translated elements with the default mappings for LaTeX and XML output can be found in section B.2.

5.6.3 Document preamble

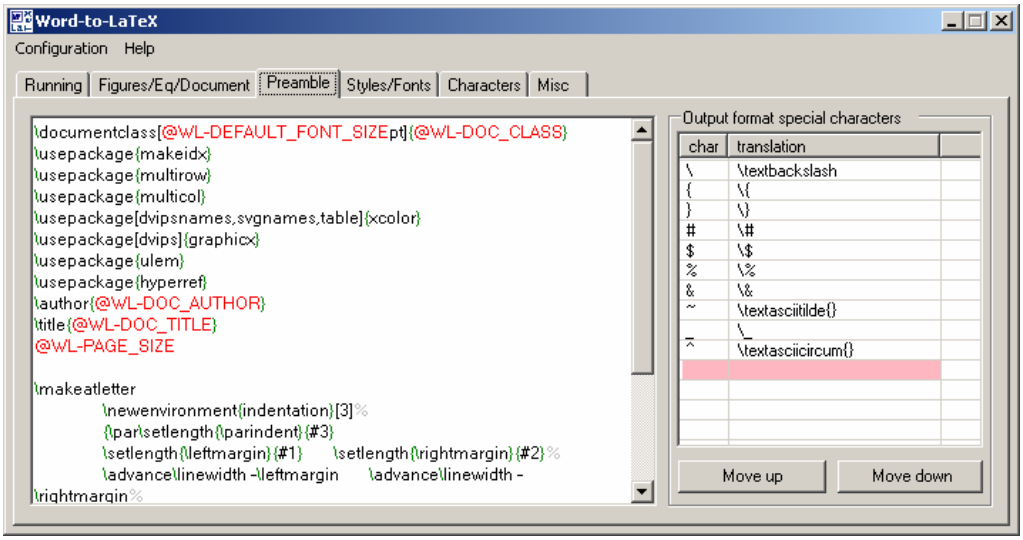


Figure 5.3: "Preamble" tab

Document preamble, inserted at the top of output files, can be easily edited in this dialog. Table 5.4 shows the list of macros that can be used in the preamble.

The translations of **Output format special characters** (e.g. "\" in LaTeX or "<" in XML) are defined in the right part of this dialog. Don't forget to fill in these characters in the right order because some special characters can be used for the translation of other special characters (e.g. "\" must be at the top for LaTeX output). New characters can be added double-clicking the pink row.

5.6.4 Special characters

Special characters are divided into groups according to their Unicode [11] positions. Each character can have a **translation** used in regular text context and a **math translation** used in math context. Currently when a character has both translations defined, the text translation is always used. If it has only a math translation, the character is inserted as a simple inline equation. If no translation is defined, the character is inserted "as is" (in UTF-8 encoding).

The math translation does not influence the conversion of equations. which is completely defined in a TDL file (see section 5.6.2 for details).

Macro	Replaced with
`@WL-DOC_CLASS`	the **Document class** option from the previous dialog
`@WL-DOC_AUTHOR`	the input document's author (retrieved from the document's properties)
`@WL-DOC_TITLE`	the input document's title (retrieved from the document's properties)
`@WL-PAGE_SIZE`	see the **Document settings** in the previous section
`@WL-DEFAULT_FONT_SIZE`	the default font size; details in section 5.6.5
`@WL-STYLE_COMMANDS`	the commands created from paragraph and character user styles, see the **Styles/Fonts** tab in section 5.6.5 for details.

Table 5.4: Document preamble macros

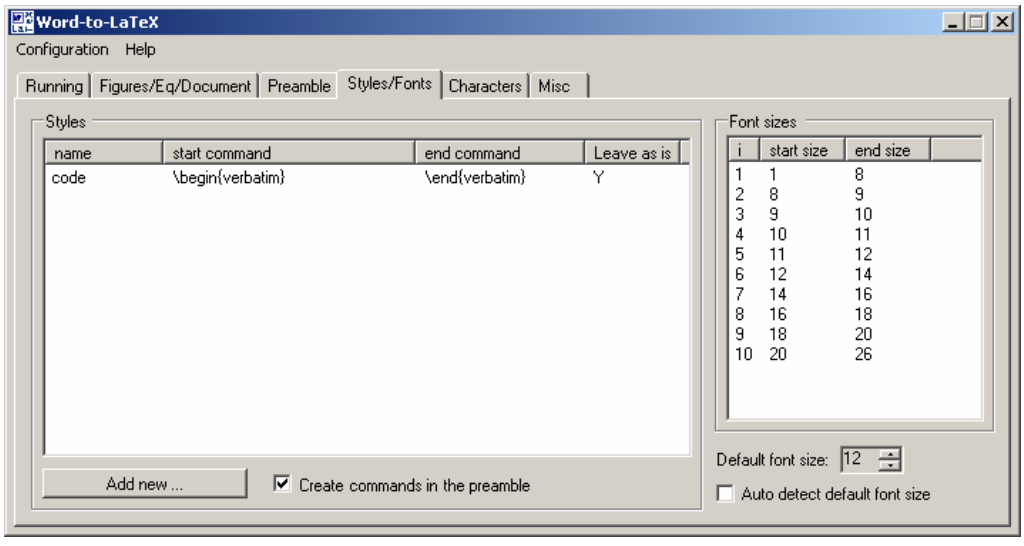


Figure 5.4: "Characters" tab

Default translations can be changed double-clicking the field you want to edit. The encoding of output files is UTF-8 which covers all national characters, so there is no need to define translations for Latin extended characters (e.g. "á") or Cyrillic ones. Just make sure that you have appropriate commands in the document preamble, for example:

```
\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}
```

5.6.5 Styles and Font sizes

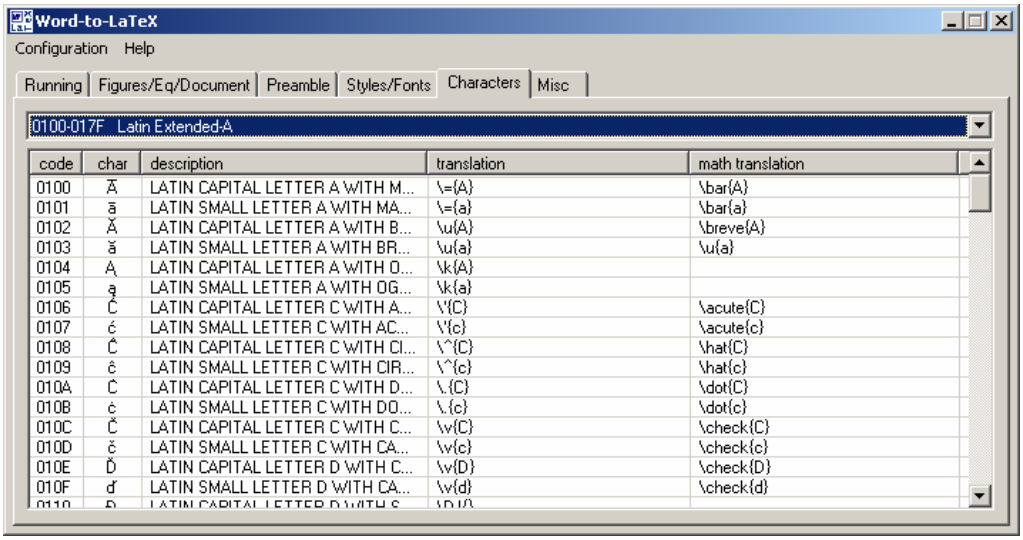


Figure 5.5: "Styles/Fonts" tab

The translations of paragraph and character user styles can be defined in this dialog. Press **Add new ...** and fill in the **name** of a style, the **start command** inserted before the text content of the style and the **end command** inserted after the text content. When you omit the definition of some style, appropriate commands will be created automatically on the basis of the style properties. *Word* built-in styles are skipped.

You can edit the list of styles double-clicking any of the fields. Write **Y** (or **N**) to the **leave as is** field if you don't want to make any changes (character translations, wrapping) in the text content of the style. It's suitable for styles that are translated to the `verbatim` environment.

Check **Create commands in the preamble** to make a special command for each style in the document preamble. It's recommended to enable this option because it makes output files much more maintainable. For example, if you have a style named "code", `\stylecode` command will be created and when you decide to change the definition of the style, you will do it only in one place.

Font sizes are split into 10 groups which are converted to the commands defined in **Translations** (see 5.6.2 for details). Each group has a point range of sizes that it covers – from the **start size** (exclusively) to the **end size** (inclusively). You can edit the default settings double-clicking the **end size** field of a group you want to change. Start sizes are counted automatically.

The portions of text that have the **Default font size** won't be marked with any command defining the font size. Therefore it's very important to have a correct value in this field to avoid a lot of unnecessary font size commands in the output file. Check **Auto detect default font size** to retrieve the default size from the *Word* built-in *Normal* style.

5.6.6 Miscellaneous options

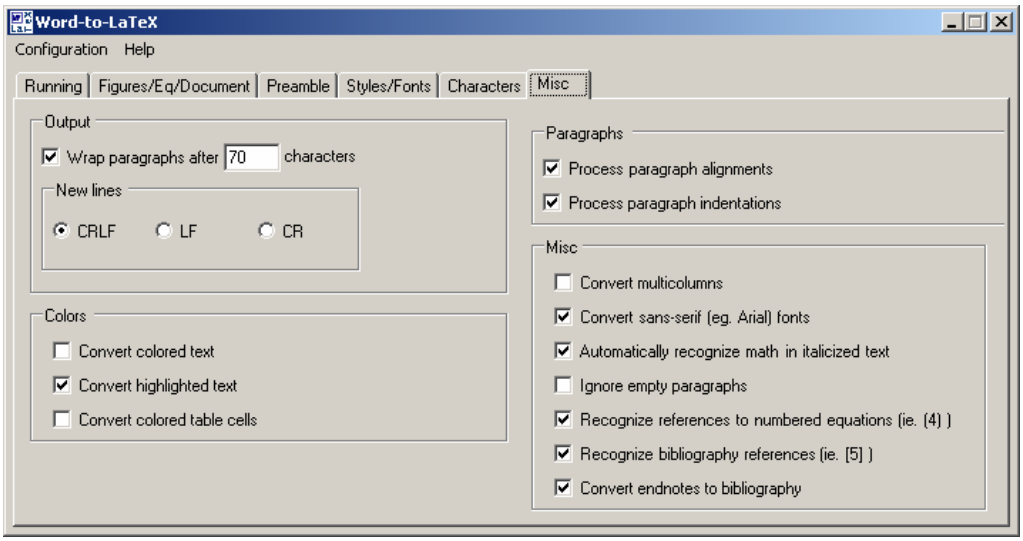


Figure 5.6: "Misc" tab

Output

Check **Wrap paragraphs** and insert an integer number to wrap the paragraphs in the output text file. The following line separators can be used in output files: **CRLF** (Windows), **LF** (Unix), **CR** (Macintosh).

Paragraphs

Check **Process paragraph alignments** and **Process paragraph indentations** to take them into account. Sometimes it's better to ignore *Word* alignments and indentations because LaTeX can make them automatically and better.

Colors

Check **Convert colored text** to convert colored portions of text using `xcolor` package. But be very careful when checking this option because it takes a lot of time to find and convert the colored text.

The same package is used when you check **Convert highlighted text** (marked with the *Word* **Highlight** tool) and **Convert colored table cells**.

When any option is unchecked, it only means that commands defining colors won't be inserted into the output file. The whole text content will be, of course, converted.

Misc

Check **Convert multicolumns** to convert multicolumn sections inserted through **Format | Columns**. Sans-serif fonts like *Arial* or *Verdana* are converted to appropriate commands only when **Convert sans-serif fonts** is checked.

Check the option **Automatically recognize math in italicized text** and simple math expressions like i or $k < 30$ will be inserted as math text instead of text in italics.

The convertor can **Recognize references to numbered equations** if they match the pattern `([1-9]+)` or `([1-9]+.[1-9]+)` (e.g. `(3.15)`). A numbered equation must be inserted on a separate line and its label must be written at the right part of the same line. Any number of white space characters between the equation and its label is allowed.

Paragraphs not containing any text won't be converted when **Ignore empty paragraphs** is checked.

Word-to-LaTeX can **Convert endnotes into bibliography** items and **Recognize bibliography references** (citations) if they match the pattern `\[[A-Za-z0-9]+\]` (e.g. `[4]` or `[Ka76]`). But if you don't use endnotes for bibliography items, you will still have to edit the bibliography section manually.

5.7 Running Word-to-LaTeX from Word

The conversion will be at least 10 times faster if you press the button on the *Word-to-LaTeX* toolbar installed directly into your *Word* application. The convertor interface is completely the same as the one described in the previous section.

If you have problems with running the convertor from *Word*, please verify that you have **Medium** or **Low** option checked in the *Word* **Tools** | **Macro** | **Security** menu.

Figure 5.7: *Word-to-LaTeX* toolbar in *Word*

5.8 Conversion to XML, XHTML, MathML

The output of the convertor completely depends on the configuration. There is no need to convert documents only to LaTeX. The `XMLConfig.xml` configuration file, stored in the *Word-to-LaTeX* directory, is used for conversion to XML [19] which is a nice intermediate format that can be easily transformed to whatever format you need. You should be familiar with XML and related technologies to understand a short overview.

The best way to insert mathematical equations into XML documents is MathML language. *Word-to-LaTeX* uses *MathType* built-in capability to export equations to MathML format.

XML format is very strict – XML files must be so-called "well-formed". Sometimes the convertor produces a file that is not well-formed, but it's never difficult to correct such a file manually.

Once we have a well-formed XML file, an XSLT style [20] can be used to transform the file into the format we need. The `html.xsl` style, located in the *Word-to-LaTeX* directory, transforms the input file to XHTML format [21] combined with CSS [22]. This style was tested with *saxon* XSLT processor.

Appendix A

Sample documents

The following pages show two documents converted with *Word-to-LaTeX*.

1. Font styles

1.1. Styles 1

Lorem ~~ipsum dolor~~ sit amet, consectetuer adipiscing elit. UT SED NISI vel justo lobortis venenatis. **Sed id risus**. Donec sollicitudin. Aenean nulla. Nam blandit, *sapien a venenatis viverra*, velit nisl mattis urna, `non luctus` sapien ante et leo. H_2O, $E = mc^2$

1.2. Styles 2

Lorem ***ipsum dolor*** sit amet[1], consectetuer adipiscing elit. Ut sed nisi vel justo lobortis venenatis. Sed id risus. Donec sollicitudin. Aenean nulla. **Nam blandit**, sapien a venenatis viverra, velit nisl mattis urna, non luctus sapien ante et leo.

2. Special characters in list

- Žluťoučký kůň úpěl ďábelské ódy.
  - $\Psi\ \Omega\ \alpha\ \zeta\ \delta$; $i \in T$; $(a,b) \notin A \times B$.

3. Paragraph indentation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut sed nisi vel justo lobortis.

4. Simple table

Blue	**Center bold**	Right
2-1	*Italics*	Pink

5. Complex table

Header			
A	a	b	B
	c	d	

[1] Lorem ipsum dolor sit amet

LaTeX output compiled to PostScript

1 Font styles

1.1 Styles 1

Lorem ~~ipsum dolor~~ sit amet, consectetuer adipiscing elit. Ut sed nisi vel justo lobortis venenatis. Sed id **risus**. Donec sollicitudin. Aenean nulla. Nam blandit, *sapien a venenatis viverra*, velit nisl mattis urna, `non luctus sapien` ante et leo. H_2O, $E = mc^2$

1.2 Styles 2

Lorem ***ipsum dolor*** sit amet[1], consectetuer adipiscing elit. Ut sed nisi vel justo lobortis venenatis. Sed id risus. Donec sollicitudin. Aenean nulla. **Nam blandit**, sapien a venenatis viverra, velit nisl mattis urna, non luctus sapien ante et leo.

2 Special characters in list

- Žluťoučký kůň úpěl ďábelské ódy.
  - Ψ Ω α ζ δ; i $\in$ T; (a,b) $\notin$ A $\times$ B.

3 Paragraph indentation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut sed nisi vel justo lobortis.

4 Simple table

Blue	**Center bold**	Right
2-1	*Italics*	Pink

5 Complex table

Header			
A	a	b	B
	c	d	

[1] Lorem ipsum dolor sit amet

Font styles

Styles 1

Lorem ~~ipsum dolor~~ sit amet, consectetuer adipiscing elit. UT SED NISI vel justo lobortis venenatis. **Sed id risus**. Donec sollicitudin. Aenean nulla. Nam blandit, *sapien a venenatis viverra* , velit nisl mattis urna, `non luctus sapien` ante et leo. H_2O, $E = mc^2$

Styles 2

Lorem ***ipsum dolor*** sit amet (Lorem ipsum dolor sit amet) , consectetuer adipiscing elit. Ut sed nisi vel justo lobortis venenatis. Sed id risus. Donec sollicitudin. Aenean nulla. **Nam blandit**, sapien a venenatis viverra, velit nisl mattis urna, non luctus sapien ante et leo.

Special characters in list

- Žluťoučký kůň úpěl ďábelské ódy.
  - $\Psi\ \Omega\ \alpha\ \zeta\ \delta;\ i \in T;\ (a,b) \notin A \times B.$

Paragraph indentation

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut sed nisi vel justo lobortis.

Simple table

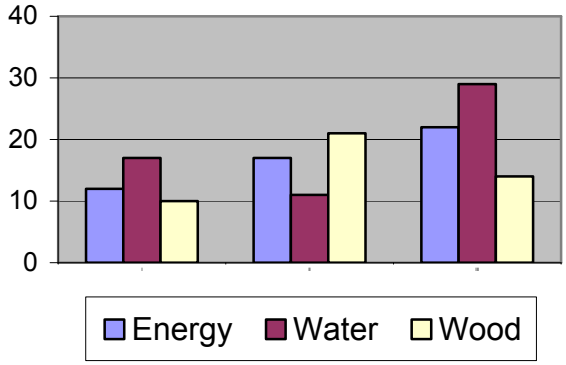
Blue	**Center bold**	Right
2-1	*Italics*	Pink

Complex table

Header			
A	a	b	B
	c	d	

Original *Word* document at the top, LaTeX output compiled to PostScript at the bottom

Bitmap image



Microsoft Excel graph

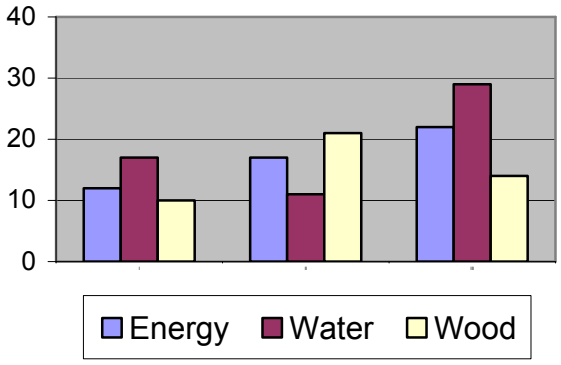
Equation editor expressions

$$D(o_i, o_j) = \sum_{k=1}^{\max(l_i, l_j)} d(o_i{}^k, o_j{}^k) \quad (1)$$

Given a set of paths X_P and a set of path contents X_{PC}, **binary relation *PPC*** $\subseteq X_P \times X_{PC}$ is defined. An $\langle e, s \rangle \in PPC$ denotes the assignment of the path $e = e_1 / e_2 / \ldots / e_k$ to the path content $s = s_1 / s_2 / \ldots / s_k$.

EQ field expression - $\frac{3}{5}$. See expression (1).

Bitmap image



Microsoft Excel graph

Equation editor expressions

$$D(o_i, o_j) = \sum_{k=1}^{\max(l_i, l_j)} d(o_i^k, o_j^k) \quad (1)$$

Given a set of paths X_P and a set of path contents X_{PC}, **binary relation *PPC*** $\subseteq X_P \times X_{PC}$ is defined. An $\langle e, s \rangle \in PPC$ denotes the assignment of the path $e = e_1/e_2/\ldots/e_k$ to the path content $s = s_1/s_2/\ldots/s_k$.

EQ field expression - $\frac{3}{5}$. See expression (1) .

Appendix B

Structure of configuration files

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns='http://kebrt.cz/word-to-latex'
      xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
    <variousOptions>
    <option name="OUTPUT_FORMAT" value="latex" />
        <option name="EQUATIONS" value="toimages" />
                    ...
    </variousOptions>
    <translationTable>
        <docElement name='FONT_BOLD'
                    start='\textbf{' end='}' />
        <docElement name='HEADING1' start='\part{' end='}' />
                    ...
    </translationTable>
    <specialChars>
        <latexChar char='\' convertTo='\textbackslash ' />
                    ...
</specialChars>
</configuration>
```

Figure B.1: Fragment of the `config.xml` configuration file

All the configuration is stored in an XML file with the `<configuration>` root element which contains three subelements:

`<variousOptions>`	various options applied during the conversion (output format, PostScript printer name, ...)
`<translationTable>`	table containing mappings between input document elements (sections, paragraphs, footnotes, and so on) and LaTeX commands
`<specialChars>`	translation mappings between special (and national) characters and LaTeX commands

B.1 Conversion options

All the options, listed in table B.1, belong to the `<variousOptions>` parent element. Each of the them is inserted into the `<option>` element with two attributes, `name` and `value`.

Option name	Description and possible values
`ONLY_IMAGES`	Convert only images and ignore text content. • `yes` × `no`
`PRINTER_NAME`	The name of a PostScript printer which is used for exporting images in EPS format. The printer driver has to be installed on your system. • e.g. `Generic Color PS`
`IMAGE_FORMAT`	The output format of images. • `eps` for EPS vector format; requires a PostScript printer • `png` for PNG bitmap format; not all the images can be exported as bitmaps
`TDL_FILENAME`	The translation file used for the conversion of equations. See the `Translators` subdirectory of your *MathType* directory for possible values (remember that *MathType* must be installed on your system to be able to convert equations). You can edit or add new files into this directory if you want to customize the conversion of equations. • e.g. `LaTeX.tdl`
`EQUATIONS`	The conversion of equations, covers *Equation Editor*, *MathType* and `EQ` fields equations. • `ignore` – do not convert • `convert` – convert using the translation file specified in the `TDL_FILENAME` option • `toimages` – convert to images
`CREATE_COMMANDS_FOR_STYLES`	The convertor will create (or not) new commands for paragraph and characters user styles in the preamble. Output text files are more maintainable if commands like `\code` are used instead of for example `\texttt`. • `yes` × `no`
`DOC_CLASS`	The `@WL-DOC_CLASS` macro used in the preamble will be replaced with the value of this option. • e.g. `article`

Table B.1: Conversion options

Option name	Description and possible values
OUTPUT_FORMAT	The format of output files. Please remember that all translations mappings described in B.2 should be set to match this output format. The convertor performs a few special actions depending on two possible values: • latex • xml
PAGE_SIZE	The @WL-PAGE_SIZE macro used in the document preamble will be replaced with the value of this option (only if the PAGE_SIZE_PROCESSING option is set to my). • e.g. a4paper
PAGE_SIZE_ PROCESSING	Specifies how the page size will be processed, possible values are: • complete – the @WL-PAGE_SIZE macro used in the document preamble will replaced with the complete page size definition matching the page size of the input document • symbolic – the convertor will try to translate the symbolic page size of the input document (e.g. A4) to an appropriate LaTeX size (e.g. letterpaper) • my – see the previous option
DEFAULT_FONT_SIZE	Defines the default font size of the input document. The portions of text having this size won't be marked with any font size command in the output file. Only integer numbers are allowed. • e.g. 12
PARAGRAPH_ ALIGNMENTS	Convert paragraph alignments. – yes × no
PARAGRAPH_ INDENTATION	Convert paragraph indentations. – yes × no
COLOR_TEXT	Use special commands for colored text. • yes × no
COLOR_BG	Use special commands for text with colored background. • yes × no
COLOR_TABLE	Use special commands for table cells with colored background. • yes × no

Table B.1: Conversion options

Option name	Description and possible values
AUTO_DETECT_ DEFAULT_FONT_SIZE	Detect the default font size of the input document automatically or not. The font size of the *Word* built-in *Normal* style will be taken as the default one if this option is set to yes. • yes × no
MULTICOLUMN	Convert multicolumn sections. • yes × no
WRAP_PARAGRAPHS	A positive value causes paragraphs to be wrapped into lines after each x characters. Any other value forces the convertor not to wrap paragraphs. • e.g. 80
NEW_LINE	Defines the line separator, possible values are: • crlf – Windows line separator • cr – Macintosh line separator • lf – Unix line separator
SANS_SERIF	Use special commands for sans-serif fonts. • yes × no
AUTO_RECOGNIZE_ MATH	Recognize math expressions written in italics (e.g. i). • yes × no
IGNORE_EMPTY_PAR	Ignore paragraphs not containing any text. • yes × no
RECOGNIZE_ NUMBERED_EQ_REF	Recognize references to numbered equations marked with labels like "(5)" or "(5.2)". • yes × no
ENDNOTES_TO_BIBLIO	Convert endnotes to bibliography items. • yes × no
RECOGNIZE_BIBLIO_ REF	Recognize in-text citations (references to bibliography items, e.g. "[4]"). – yes × no
FONT_SIZE[1-10]	These options define ranges for each converted font size group. The range for the i-th group is from FONT_SIZE(i-1)+1 to FONT_SIZE(i) (inclusive). The first group (FONT_SIZE1) starts with the size 1. Only integer numbers are allowed. • e.g. 11 for the FONT_SIZE4 option and 12 for the FONT_SIZE5 option when the default font size is 12

Table B.1: Conversion options

B.2 Conversion mappings

Table B.2 shows the complete list of conversion mappings between input document elements (sections, paragraphs, lists, and so on) and *Word-to-LaTeX*. Each mapping has a start command (`S:`) which is inserted before the element and most of them have also an end command (`E:`) inserted after the element. Some elements like tabulators doesn't have any content, others hold some kind of content (text, equation, another element) which is inserted between the start and end command.

Names of macros that are specific to each element begin with "`#`", macros common to all elements begin with "`@`".

- `@WL-NL` new line
- `@WL-TAB` tabulator

Table B.2 also contains the default mappings for LaTeX and XML output. When `E:` is omitted, the end command is always ignored by the convertor, "—" stands for the empty translation command.

`FONT_BOLD`	bold font
`S: \textbf{`	
`E: }`	
`S: <font type="bold">`	
`E: </font>`	
`FONT_ITALIC`	italic font
`S: \textit{`	
`E: }`	
`S: <font type="italic">`	
`E: </font>`	
`FONT_SMALLCAPS`	small caps font
`S: \textsc{`	
`E: }`	
`S: <font type="smallcaps">`	
`E: </font>`	
`FONT_HIDDEN`	hidden font
`S: @WL-NL%`	
`E: @WL-NL`	
`S: <font type="hidden">`	
`E: </font>`	

Table B.2: Conversion mappings

FONT_SUBSCRIPT	subscript font
S: $_{	
E: }$	
S: 	
E: 	
FONT_SUPERSCRIPT	superscript font
S: $^{	
E: }$	
S: 	
E: 	
FONT_COURIER	courier font (e.g. Courier, Courier New)
S: \texttt{	
E: }	
S: 	
E: 	
FONT_UPPERCASE	uppercase font
S: \uppercase{	
E: }	
S: 	
E: 	
FONT_UNDERLINE	underlined font
S: \uline{	
E: }	
S: 	
E: 	
FONT_DOUBLE_UNDERLINE	double-underlined font
S: \uuline{	
E: }	
S: 	
E: 	
FONT_WAVE_UNDERLINE	wavy-underlined font
S: \uwave{	
E: }	
S: 	
E: 	

Table B.2: Conversion mappings

FONT_STRIKE	strikethrough font
S: \sout{	
E: }	
S: 	
E: 	
FONT_SANS_SERIF	sans-serif font (e.g. Arial, Verdana)
S: \textsf{	
E: }	
S: 	
E: 	
FONT_SIZE1	font size (group 1)
S: {\tiny	
E: }	
S: <font-size value="1">	
E: </font-size>	
FONT_SIZE2	font size (group 2)
S: {\scriptsize	
E: }	
S: <font-size value="2">	
E: </font-size>	
FONT_SIZE3	font size (group 3)
S: {\footnotesize	
E: }	
S: <font-size value="3">	
E: </font-size>	
FONT_SIZE4	font size (group 4)
S: {\small	
E: }	
S: <font-size value="4">	
E: </font-size>	
FONT_SIZE5	font size (group 5)
S: {\normalsize	
E: }	
S: <font-size value="5">	
E: </font-size>	

Table B.2: Conversion mappings

FONT_SIZE6	font size (group 6)
S: {\large	
E: }	
S: <font-size value="6">	
E: </font-size>	
FONT_SIZE7	font size (group 7)
S: {\Large	
E: }	
S: <font-size value="7">	
E: </font-size>	
FONT_SIZE8	font size (group 8)
S: {\LARGE	
E: }	
S: <font-size value="8">	
E: </font-size>	
FONT_SIZE9	font size (group 9)
S: {\huge	
E: }	
S: <font-size value="9">	
E: </font-size>	
FONT_SIZE10	font size (group 10)
S: {\Huge	
E: }	
S: <font-size value="10">	
E: </font-size>	
HEADING1	heading (level 1); headings have to be marked with the *Word* built-in styles; they can be defined up to level 9
S: \section{	
E: }	
S: <heading level="1">	
E: </heading>	
HEADING2	heading (level 2)
S: \subsection{	
E: }	
S: <heading level="2">	
E: </heading>	

Table B.2: Conversion mappings

HEADING3	heading (level 3)
S: \subsubsection{	
E: }	
S: <heading level="3">	
E: </heading>	
ALIGN_CENTER	paragraph alignment – centered
S: \begin{center}@WL-NL	
E: @WL-NL\end{center}	
S: <align type="center" />	
E: —	
ALIGN_LEFT	paragraph alignment – left
S: {\raggedright@WL-NL	
E: @WL-NL}	
S: <align type="left" />	
E: —	
ALIGN_RIGHT	paragraph alignment – right
S: {\raggedleft@WL-NL	
E: @WL-NL}	
S: <align type="right" />	
E: —	
TABLE_ALIGN_CENTER	table paragraph alignment – centered
• #WIDTH	table cell width (in points)
S: \parbox{#WIDTHpt}{\centering	
E: }	
S: <align type="center" />	
E: —	
TABLE_ALIGN_LEFT	table paragraph alignment – left
• #WIDTH	table cell width (in points)
S: \parbox{#WIDTHpt}{\raggedright	
E: }	
S: <align type="left" />	
E: —	

Table B.2: Conversion mappings

TABLE_ALIGN_RIGHT	table paragraph alignment – right
• #WIDTH	table cell width (in points)
S: \parbox{#WIDTHpt}{\raggedleft	
E: }	
S: <align type="right" />	
E: —	
FOOTNOTE	footnote
S: \footnote{	
E: }	
S: <footnote>	
E: </footnote>	
PAGE_BREAK	page break
S: \pagebreak{}@WL-NL@WL-NL	
S: <pagebreak />	
EQUATION_INLINE	inline equation
S: \begin{math}	
E: \end{math}	
S: <equation type="inline">	
E: </equation>	
EQUATION_NUMBERED	numbered equation
• #ORIG_LABEL	original equation label retrieved from the input document
S: \begin{equation}	
E: @WL-NL%#ORIG_LABEL@WL-NL\end{equation}	
S: <equation type="numbered" origlabel="#ORIG_LABEL">	
E: </equation>	
EQUATION_LABEL	equation label inserted into the EQUATION_NUMEBERED element
• #NAME	auto-generated label (auto-incrementing counter is used)
S: \label{#NAME}	
S: <label name="#NAME"/>	

Table B.2: Conversion mappings

EQUATION_OUTLINE	equation displayed on a separate line
S: \begin{displaymath}	
E: \end{displaymath}	
S: <equation type="outline">	
E: </equation>	
INDEX_ENTRY	index entry (*Word* XE field)
S: \index{	
E: }	
S: <index-entry>	
E: </index-entry>	
INDEX	index (*Word* INDEX field), LaTeX generates the whole index automatically
S: \printindex	
S: <printindex />	
IMAGE_COMMAND	image
• #WIDTH	image width (in points)
• #FILENAME	auto-generated image filename (e.g. img1.eps)
• #TITLE	image title (if present)
S: \includegraphics[width=#WIDTHpt]{#FILENAME}@WL-NL	
S: <image width="#WIDTH" src="#FILENAME" title="#TITLE" />	
IMAGE_CONTAINER	image container (used when the image has a title)
S: \begin{figure}[h]@WL-NL	
E: \end{figure}	
S: —	
E: —	
IMAGE_TITLE	image title inserted into the IMAGE_CONTAINER element
• #TITLE	title
S: \caption{#TITLE}	
S: —	
TOC	table of contents (*Word* TOC field), LaTeX generates the table of contents automatically as well as *Word*
S: \tableofcontents	
S: <table-of-contents />	

Table B.2: Conversion mappings

HYPERLINK	hyperlink
• #HREF	hyperlink target; the macro can be used also in the end command
S: \href{#HREF}{	
E: }	
S: <link href="#HREF">	
E: </link>	
SPECIAL_COMMAND	LaTeX command(s) inserted into the document through the *Word* PRIVATE field whose content must begin with the case-insensitive string latex:, such a field may look like this: PRIVATE LaTeX: \indent (\indent will be inserted between the start and end command)
S: —	
E: —	
S: —	
E: —	
REFERENCE	bookmark reference
• #NAME	name of the bookmark that is being referenced
S: \ref{#NAME}	
S: <reference name="#NAME" />	
MATH_REFERENCE	equation reference; the *Word* hard-coded reference (e.g. "(3)") will be the content of this element
• #NAME	name of the equation that is being referenced, it is generated for each numbered equation in the document (e.g. "eq3").
S: (\ref{#NAME})@WL-NL%	
E: @WL-NL	
S: <math-reference name="#NAME">	
E: </math-reference>	
NOTE_REFERENCE	note reference; currently only endnotes are supported
• #NAME	name of the note (typically number) that is being referenced
S: \cite{ref#NAME}	
S: <note-reference name="#NAME" />	

Table B.2: Conversion mappings

BIBLIO_REFERENCE	reference to a bibliography item ("citation"); the *Word* hard-coded citation (e.g. "[Ka75]") will be the content of this element
• #NAME	name of the bibitem (e.g. "Ka75")
S: \cite{ref#NAME}@WL-NL%	
E: @WL-NL	
S: <biblio-reference name="#NAME">	
E: </biblio-reference>	
PAGE_REFERENCE	page reference
• #NAME	name of the bookmark that is being referenced
S: \pageref{#NAME}	
BOOKMARK_LABEL	bookmark
• #NAME	name of the bookmark
S: \label{#NAME}	
S: <bookmark name="#NAME" />	
STYLE	paragraph or character user style
• #NAME	name of the style; all numbers in the name are replaced with words (e.g. "1" → "One")
S: \#NAME{	
E: }	
S: <style name="#NAME">	
E: </style>	
STYLE_DEFINITION	container for a single user style definition; commands describing the style will be inserted into
• #NAME	name of the user style
S: \newcommand{\#NAME}[1]{	
E: }	
S: <style-definition name="#NAME">	
E: </style-definition>	
DOCUMENT_BODY	document body
S: \begin{document}@WL-NL	
E: \end{document}	
S: <body>	
E: </body></document>	

Table B.2: Conversion mappings

LIST_ENUMERATE	enumerated list
S: \begin{enumerate}@WL-NL	
E: \end{enumerate}@WL-NL@WL-NL	
S: @WL-NL<list type="enumerate">	
E: </list>@WL-NL	
LIST_ITEMIZE	**itemized list**
S: \begin{itemize}@WL-NL	
E: \end{itemize}@WL-NL@WL-NL	
S: @WL-NL<list type="itemize">	
E: </list>@WL-NL	
LIST_ITEM	**list item**
S: @WL-TAB\item	
E: —	
S: <list-item>	
E: </list-item>@WL-NL	
PARAGRAPH	**common paragraph**
S: —	
E: @WL-NL@WL-NL	
S: @WL-NL<para>	
E: </para>@WL-NL	
TABLE_PARAGRAPH	**paragraph in a table**
S: @WL-NL	
E: @WL-NL	
S: @WL-NL<table-para>	
E: </table-para>@WL-NL	
LIST_PARAGRAPH	**paragraph in a list**
S: —	
E: @WL-NL	
S: <list-para>	
E: </list-para>	
LINE_BREAK	**line break**
S: @WL-NL\\@WL-NL	
S: <linebreak />	
TAB	**tabulator**
S: \hspace{15pt}	
S: <tab />	

Table B.2: Conversion mappings

TABLE_CELL	table cell
• #WIDTH	cell width
S: &	
E: —	
S: <table-cell width="#WIDTH">	
E: </table-cell>	
TABLE_ROW	table row
S: —	
E: \\@WL-NL	
S: <table-row>	
E: </table-row>	
TABLE	table
• #TITLE	title of the table
S: @WL-NL\vspace{3pt} \noindent@WL-NL\begin{tabular}	
E: \end{tabular}\\@WL-NL\vspace{2pt}@WL-NL	
S: @WL-NL<table title="#TITLE">	
E: </table>@WL-NL	
TABLE_CONTAINER	table container (used when the table has a title)
S: @WL-NL\begin{table}[h]	
E: \end{table}@WL-NL	
S: —	
E: —	
TABLE_TITLE	table title inserted into the TABLE_CONTAINER element
• #TITLE	title
S: \caption{#TITLE}	
S: —	
TABLE_MULTIROW	table cell with merged rows
• #ROWS	number of merged rows in the cell
S: \multirow{#ROWS}{*}{	
E: }	
S: <table-multirow-cell multi="#ROWS" />	
E: —	

Table B.2: Conversion mappings

TABLE_CELL_COLOR	command for the colored background of table cells; the #COLOR macro in the next element (TABLE_MULTI_COLUMN) will be replaced with this command
• #COLOR	background color in HTML notation (e.g. FF0000)
S: >{\columncolor[HTML]{#COLOR}}	
S: color="#COLOR"	
TABLE_MULTICOLUMN	table cell with merged columns
• #COLS	number of merged columns
• #LEFT_BORDER	"|" if the cell has a left border
• #RIGHT_BORDER	"|" if the cell has a right border
• #COLOR	see the previous element
• #ALIGN	cell content alignment; l (left), r (right), c (center)
S: \multicolumn{#COLS}{#LEFT_BORDER#COLOR#ALIGN#RIGHT_BORDER}{	
E: }	
S: <table-cell multi="#COLS" left-border="#LEFT_BORDER" right-border="#RIGHT_BORDER" align="#ALIGN" width="#WIDTH" #COLOR>	
E: </table-cell>	
PAR_INDENT	paragraph indentation
• #LEFT_INDENT	left indentation (in points)
• #RIGHT_INDENT	right indentation (in points)
• #FIRST_LINE_INDENT	first line indentation (in points)
S: \begin{indentation}{#LEFT_INDENTpt}{#RIGHT_INDENTpt} {#FIRST_LINE_INDENTpt}@WL-NL	
E: @WL-NL\end{indentation}	
S: @WL-NL<par-indent left="#LEFT_INDENT" right="#RIGHT_INDENT" first-line="#FIRST_LINE_INDENT" />@WL-NL	
E: —	
MULTICOLUMN	multicolumn section
• #COLS	number of columns in the section
S: \begin{multicols}{#COLS}	
E: \end{multicols}	
S: <multicol count="#COLS">	
E: </multicol>	

Table B.2: Conversion mappings

COLOR_TEXT	colored text
• #COLOR	color in HTML notation (e.g. FF0000)
S: \textcolor[HTML]{#COLOR}{	
E: }	
S: <font-color color="#COLOR">	
E: </font-color>	
COLOR_BG	text with colored background
• #COLOR	color in HTML notation (e.g. FF0000)
S: \colorbox[HTML]{#COLOR}{	
E: }	
S: <font-background color="#COLOR">	
E: </font-background>	
ENDNOTES_SECTION	container for endnotes, can be used for inserting the bibliography
S: \begin{thebibliography}{99}@WL-NL	
E: \end{thebibliography}@WL-NL	
S: <bibliography>	
E: </bibliography>	
ENDNOTE	endnote, this translation is used in the ENDNOTES_SECTION context, suitable for inserting a single bibliography item
• #NUMBER	number of the endnote
S: @WL-TAB\bibitem[#NUMBER]{ref#NUMBER}	
E: @WL-NL	
S: @WL-TAB<bib-item name="#NUMBER">	
E: </bib-item>	
ENDNOTE_REFERENCE	endnote, this translation is used at the endnote's insertion point
• #NUMBER	number of the endnote
• #CONTENT	endnote's text content (can be used when translating endnotes to footnotes)
S: \cite{ref#NAME}	
S: <endnote-reference name="#NUMBER" />	

Table B.2: Conversion mappings

COLOR_BG_AND_BORDER	text with colored border and background
• #BORDER_COLOR	border color, in HTML notation (e.g. FF0000)
• #COLOR	text color, dtto
S: \fcolorbox[HTML]{#BORDER_COLOR}[HTML]{#COLOR}{	
E: }	
S: <box border-color="#BORDER_COLOR" background-color="#COLOR">	
E: </box>	
COLOR_BORDER	colored border around text
• #BORDER_COLOR	border color, in HTML notation (e.g. FF0000)
S: \fcolorbox[HTML]{#BORDER_COLOR}[HTML]{FFFFFF}{	
E: }	
S: <box border-color="#BORDER_COLOR">	
E: </box>	
BORDER	black border around text
S: \fbox{	
E: }	
S: <box>	
E: </box>	

Table B.2: Conversion mappings

B.3 Special characters

The configuration of special characters is enclosed in the `<specialChars>` element. `<latexChar>` elements are used for defining characters that have a special meaning in the output format. They must be written in a correct order because one special character can be used for translating another special character which is illustrated in the following example.

```
<latexChar char='\' convertTo='\textbackslash ' />
<latexChar char='{' convertTo='\{' />
```

All the other special and national characters are defined in `<char>` elements. The `code` attribute contains the Unicode [11] number of each character. The details about the common context translation (`convertTo` attribute) and the math context translation (`mathConvertTo` attribute) can be found in section 5.6.4. A short example follows.

```
<char code="010C" convertTo="\v{C}" mathConvertTo="\check{C}" />
<char code="010D" convertTo="\v{c}" mathConvertTo="\check{c}" />
```

Bibliography

[1] Allin Cottrell. *Word Processors: Stupid and Inefficient*, `http://www.ecn.wfu.edu/~cottrell/wp.html`

[2] Donald E. Knuth. *The TEXbook*, Volume A of *Computers and Typesetting*, Addison-Wesley Publishing Company (1984), ISBN: 0-201-13448-9.

[3] Tobias Oetiker. *The Not So Short Introduction to LaTeX 2ε*, `http://people.ee.ethz.ch/~oetiker/`

[4] Marion Neubauer. *Conversion from WORD/WordPerfect to LaTeX*, MAPS 14, 1995, 120-124, `http://www.ntg.nl/maps/maps14.html`

[5] Jesse Liberty. *Programming C#, Second Edition*, O'Reilly (2002), ISBN: 0-596-00309-9.

[6] Ben Albahari, Peter Drayton, Brad Merrill. *C# Essentials, Second Edition*, O'Reilly (2001), ISBN: 0-596-00315-3.

[7] MSDN Library. *Word Object Model Overview*, `http://msdn2.microsoft.com/en-US/library/kw65a0we(VS.80).aspx`

[8] Julianne Sharer, Arthur Einhorn. *Word Object Model: The Definitive Reference*, O'Reilly (2001), ISBN 1-56592-430-4.

[9] MathType Software Development Kit, `http://www.dessci.com/en/reference/sdk/`

[10] Dale Rogersion. *Inside COM*, Microsoft Press (1997), ISBN: 1572313498.

[11] *Unicode Home Page*, `http://www.unicode.org/`

[12] Wilfried Hennings. *Convertors from PC Textprocessors to LaTeX*, `http://www.tug.org/utilities/texconv/pctotex.html`

[13] *wsW2LTX convertor*, `http://www.winshell.de/`

[14] *Antiword*, `http://www.winfield.demon.nl/`

[15] *GrindEQ*, `http://www.grindeq.com/`

[16] *the Comprehensive TEX Archive Network*, `http://www.ctan.org/`

[17] *rtf2latex2e*, `http://sourceforge.net/projects/rtf2latex2e/`

[18] *Word2TEX*, http://www.chikrii.com/

[19] *Extensible Markup Language (XML)*, http://www.w3.org/XML/

[20] *XSL Transformations (XSLT)*, http://www.w3.org/TR/xslt

[21] *XHTML 1.0 The Extensible HyperText Markup Language*, http://www.w3.org/TR/xhtml1/

[22] *Cascading Style Sheets*, http://www.w3.org/Style/CSS/