rtf2LaTeX 2$_\varepsilon$ Documentation
version 0.22 beta

Ujwal S. Sathyam
setlur@bigfoot.com

26. April 1999

1 Introduction

rtf2LaTeX 2$_\varepsilon$ is an RTF to LaTeXconverter written using Paul DuBois' RTF reader framework. It takes as its input RTF files produced by Microsoft Word and comparable word processors and generates a TeX-able "**.tex**" file. It has the capability to handle fairly complex RTF files containing figures, tables, and equations to some extent. **rtf2LaTeX 2$_\varepsilon$** is written using standard *C* and should compile on any platform supporting a *C* compiler. I have tested it on the Macintosh, Linux (Intel), Linux (PowerPC), Windows 95/NT, and Solaris.

This is not the first attempt at rtf2latex. Several souls have taken a shot at it to varying degrees of success. Some handled equations, some worked on figures fairly well, and some did neither. Unfortunately, when I went hunting for a converter, the ones I found were at least four years old and wouldn't work on my RTF file. RTF had obviously moved on. That was six months ago. I then decided to write a converter that would handle the latest RTF specification. Nights of furious coding followed by months of complete neglect finally resulted the latest incarnation of **rtf2LaTeX 2$_\varepsilon$**. I have decided that it is ready for public viewing and maybe even consumption since it finally succeeded my converting initial RTF file.

rtf2LaTeX 2$_\varepsilon$ uses the generic RTF reader framework by Paul DuBois. The framework is a general purpose tool for processing RTF files and may be configured in a well-defined manner to allow it to be used with a variety of writers generating different output formats. This provides a method for generating RTF-to-XXX translators. Essentially, **rtf2LaTeX 2$_\varepsilon$** provides the LaTeX 2$_\varepsilon$writer code to the RTF reader.

What you will get

If you expect a WYSIWYG reproduction of your RTF file, you may be disappointed. My main concerns have been translating the essential features of the RTF file such as character, figures, tables, and equations (as pictures). I have largely ignored visual formatting such as ruler positions, tabs (until I figure out a good way of doing this), paragraph indentations, and other fluff. I have always expected the output LaTeX 2εfile to require manual editing to put the finishing touches. I just want to make that task a little easier. In my opinion, expecting a WYSIWYG reproduction is not practical and misses the point entirely.

2 Installation and Use

2.1 Macintosh

The Macintosh distribution includes a PPC drag-and-drop application on which you can drop multiple RTF files. The output LaTeXfiles will be created in the same directory as the input RTF files. If Mac users want to build their own binaries, I have included the CodeWarrior project file (for version Pro 1). To compile, you will need to download the DropUnix1.3 application framework which is available at http:/www.zenspider.com. Then you will need to change the access path in the CodeWarrior project settings to point to the location of DropUnix on your machine.

2.2 Unix

Unix users will need to compile the binary. The sources are in the *sources* directory, and there is a Makefile. To start building the **rtf2LaTeX 2ε** binary, type:
make rtfprep
This will build a program called **rtfprep**. It basically generates some files needed by **rtf2LaTeX 2ε**. After building **rtfprep**, run it by typing:
./rtfprep
This will generate three header files and one look-up table. You are now ready to build **rtf2LaTeX 2ε** by typing:
make
To clean up, type:
make clean
To install, type:

make install
The *install* target just copies the binary and the auxiliary files to the install directory that is specified in the Makefile. You can change the directory into which rtf2latex2e is installed by editing the variable INSTALL_DIR at the top of the Makefile. Default install directory is /usr/rtf2latex2e. You may need to become superuser to install into that directory. If you do not have superuser previleges, you can change the INSTALL_DIR to somewhere in your home directory, say $(HOME)/rtf2latex2e.

Finally, you will need to set the environment variable RTF2LATEX2E_DIR from within your shell. The variable has to point to the directory into which rtf2latex2e was installed. You can set the variable using
export RTF2LATEX2E_DIR=directory (bash) or
setenv RTF2LATEX2E_DIR=directory (csh)
in your .bashrc or .login file, whichever is read by your shell. It is also convinient to add the rtf2latex2e directory to your search PATH or create a symbolic link to the binary in your /usr/bin directory.

To run the program, type:
rtf2latex2e $< rtfFileName >$
If the file name contains spaces, enclose the path in double quotes. No command line options are supported yet. I come from the Mac world, and I am not used to them. When I think of some useful options, I will provide for command line options.

2.3 Windows

Windows users get precompiled binaries of **rtf2LaTeX 2$_\varepsilon$** and **rtfprep** to be run from the MS-DOS prompt. If anyone needs to recompile either program, I have included CodeWarrior Pro 4 project files for both **rtf2LaTeX 2$_\varepsilon$** and **rtfprep**.

You will need to set the environment variable RTF2LATEX2E_DIR from within DOS. The variable has to point to the directory into which rtf2latex2e was installed. You can set the variable using
SET RTF2LATEX2E_DIR=directory
in your AUTOEXEC.BAT file.

It is also convinient to add the rtf2latex2e directory to your search PATH in the AUTOEXEC.BAT file.

Important

The character set map files (all the *gen* and *sym* files), the output map file TEX-map, and the RTF control word look-up table *rtf-ctrl* need to reside in the same directory as the **rtf2LaTeX 2$_\varepsilon$** binary.

rtfprep is an auxiliary program. This was written by Paul DuBois and modified by me: its task is to read the a list of RTF control words from a file *rtf-controls* and a list of standard character names from *standard-names* and generate the look-up table *rtf-ctrl* along with three header files. The header files (*rtf-ctrldef.h*, *rtf-namedef.h*, and *stdcharnames.h*) are required to compile **rtf2LaTeX 2$_\varepsilon$** , and the look-up table *rtf-ctrl* is required at run time. You should not need to run rtfprep ever again after the first time unless you are adding control words or standard character names. If you do, you will need to recompile **rtf2LaTeX 2$_\varepsilon$**. If you add new RTF control words and run **rtfprep**, make sure to move the newly generated *rtf-ctrl* into the same folder that contains **rtf2LaTeX 2$_\varepsilon$**. In both the Mac and Windows distributions, the rtfprep binary is in the sources folder.

The r2l-pref preference file

Based on the feedback from some early testers, I have added a feature in version 0.21 to read a preference file r2l-pref. In it, you can choose among options to ignore ruler settings and to set paper size. Ignoring ruler settings will result in the loss of some visual formatting, but it will also result in cleaner LaTeX 2$_\varepsilon$code. You also have an option to ignore color information. Finally, you can specify an encoding style for use with the inputenc package. This will cause rtf2latex2e to directly insert characters between 128-255 into the LaTeX 2$_\varepsilon$file.

3 Features

rtf2LaTeX 2$_\varepsilon$ is designed to convert journal articles, reports, and letters written in Microsoft Word. That means I would like it to handle the following:

- **Text Style:** some amount of stylized text like color, **bold**, *italic*, underlined, and relative size like small, normal, big, very big, and large. This is a little weak in older RTF files as the older RTF spec is a little crappier than the newer one. All other font information is disregarded, as TeX can do better anyway.

- **Figures:** Right now **rtf2LaTeX 2$_\varepsilon$** can read figures of format PICT, WMF, PNG, and JPEG embedded into RTF files. These are the most common formats encountered in RTF files. When rtf2LaTeXe encounters an embedded figure, it reads out the figure into a separate file. The output format of the figure is the same as the format it is embedded in. You may need to convert the figure to format appropriate for inserting into a LaTeX file, usually EPS. Figures within tables are read but not output for fear of messing up the LaTeX source. I will get to that some day.

  In version 0.22, I have added internal conversion of embedded JPEG files to EPS. The code was adapted from the program Thomas Merz's jpeg2ps program (with his permission). The conversion is equivalent to running jpeg2ps with the "-h" option, ie. hex encoding is used.

- **Equations:** The most common source of the RTF file is Microsoft Word. Equations in Word are created in Equation Editor, and when saved into an RTF file, the equation is embedded as an OLE object. Unfortunately, decoding OLE objects is beyond the scope of my skills right now. I hope to tackle it somehow in the future. Fortunately, MS Word also embeds the equation as a picture for older RTF readers. **rtf2LaTeX 2$_\varepsilon$** reads that picture and outputs the equation as a picture file. Not perfect, but it will have to do for now. At least, you can see what the equation looks like and re-implement it in LaTeX 2$_\varepsilon$.

- **Tables:** Yeah, it does tables!! However, this is the weakest link in the chain and the messiest part of the code. This is largely due to the fact that RTF does not have a separate 'Table' group. It is also due to the fact that TeX likes to know in advance the number of columns in the table, and RTF does not tell us that. I spent a lot of time to support tables to this extent. A lot of the test files have tables in them. To get an idea of the type of tables that **rtf2LaTeX 2$_\varepsilon$** can handle, take a look at **table1.rtf**, **Script.rtf**, and **RTF-Spec.rtf**. Some test files that break the program are also included in the directory *"test files not working"*. Not surprisingly, the program messes up in a table. I use longtable.sty package for table handling to take care of tables that span several pages.

- **Paragraph Style:** I care for alignment issues like centering, left, and right justification. Useful in letters. All other visual formatting like indentation is currently ignored until I figure out how to translate RTF's paragraph syntax into appropriate LateX commands/environments.

- **Character mapping:** Character mapping is not complete yet. Most, but not all, Greek and math symbols are supported by referencing character set maps and the output map file "TEX-map". I plan to support Unicode through binary search of an external file to which additions can be made, much like the external rtf-ctrl file that the RTF reader uses. This should take care of most symbols we care about. Check out the various test files with *Symbol* in their names to see how well math and greek symbols are handled. Some of those files were encoded by the RTF writer using the ANSI character set map, and some with the Macintosh character set map. When you run these files through **rtf2LaTeX 2$_\varepsilon$**, you will likely get warning messages about unknown characters. The reason I have not mapped those character is simply that I don't know how to represent them in TEX. Quite embarrassing, huh? But they are also quite arcane characters that I have never seen in any document. I will work on beefing up this portion of the code. Also, check out **math.rtf**: it has quite a bit of math in it. There is some issue with mapping of accented characters and some math symbols (ASCII range > 127) for RTF files that use the ansi character set map. RTF seems to have changed its ansi encoding somewhere between Word 5.1 and Word 97/98. Even the new versions of Word themselves (Word 97/98) do not correctly interpret some characters (beyond ASCII range 127) from RTF files produced by older versions (say Word 5.1). I have therefore included in the directory "for-older-rtf-files" the files ansi-gen and ansi-sym for older verions of RTF files. Experiment with your RTF file to determine whether you need the new set or the old set. Be sure to safely back up the versions you are not using for later use. At some point, I will write code to make the reader dynamically read the appropriate character map files. RTF files with Mac character encoding do not seem to suffer from this malaise.

  In version 0.22, I have added a feature to use the inputenc package. The type of encoding is specified in the r2l-pref preference file. Specifying an encoding type causes rtf2latex2e to directly insert into the tex file characters between 128–255 that are defined by the $<$ *encoding* $>$ *.def* file.

- **Footnotes:** It was quite simple to add footnote support. I would like to support footnotes in tables too, but TEXis not cooperating even though I am using the longtable package for tables. I will look into it further.

Features I would like to support in future versions are:

- Unicode: This should really get rid of the need for different character set maps. Word 98 on the Mac already puts out Unicode.
- Lists

4 Test files

Along with the rtf2latex2e distribution, you can also download a set of test files to see how the program behaves. These test files are in a tarred gzipped archive in the same place where you downloaded the rtf2latex2e distribution. "*RTF-test-files*" contains several RTF files that have been successfully tested on **rtf2LaTeX 2$_\varepsilon$**. By success, I mean that **rtf2LaTeX 2$_\varepsilon$** processes the RTF file without any problems (except maybe giving a few warnings) and produces a ".tex" file that is LaTeX 2$_\varepsilon$-able!! It does not mean that the TeXoutput file will look exactly the same as the RTF input file. In fact, most of the time, it will not. Some features like I do not care to convert, others like Unicode support will be implemented in future versions.

The distribution also has a directory called "*test-files-not-working*". This contains files that break the **rtf2LaTeX 2$_\varepsilon$** program. Either, the program get stuck while processing the RTF file, or it produces non-LaTeX 2$_\varepsilon$-able output files. I have included this so that you can get idea what **rtf2LaTeX 2$_\varepsilon$** can and can not do. Hopefully, I can resolve these issues soon.

5 RTF Translator Architecture

In the following sections, I will attempt to provide documentation for people who wish to correct/modify/improve the code. I shall be borrowing heavily from Paul's documentation of the reader. For detailed documentation on the RTF reader framework, read the files **rtfReader.rtf** and **rtf.Arch.rtf** in the RTFLaTeX 2$_\varepsilon$test files directory (or you can run the files through **rtf2LaTeX 2$_\varepsilon$** and typeset the resulting **rtfReader.tex** files!).

There are three components to an RTF translator: reader code, writer code, and driver code. These break down as follows:

- **reader:** Responsible for peeling tokens out of the input stream, classifying them, and causing the writer to process them.
- **writer:** Responsible for translating tokens from the input stream into the required output format. This is my contribution.

- **driver:** Responsible for making sure the reader and writer are initialized, and for calling the reader, to cause translation to occur. The function **main( )** resides here.

This architecture allows the reader to remain constant, so that different translators can be built by supplying different writer and driver code. Also, for a given translator, the reader and writer remain constant and the translator can be ported to different types of systems by supplying system-specific driver code.

In practice, to build a new translator, you supply a main() function and the writer code, and link in the RTF reader. **main( )** includes the driver code and is responsible to see that the following are done:

- Determine which files are to be translated
- Configure the reader, which may involve:
  - Reset the input stream if necessary
  - Configure other reader behavior, such as whether or not to process the font and color tables internally
  - Install writer callbacks into the reader so it knows what functions to call when various kinds of tokens occur
- Initialize the writer
- Call the reader to process the input stream
- Terminate the writer

6 Reader Operation

Each time a token is read, several global variables are set:

- **rtfClass:** token class
- **rtfMajor:** token major number
- **rtfMinor:** token minor number
- **rtfParam:** token parameter value
- **rtfTextBuf:** token text
- **rtfTextLen:** length of token (including parameter text)

Tokens are classified using up to three numbers: token class, and major and minor numbers. The major and minor numbers may be meaningless depending on the kind of token.

The class number can be:

- **rtfUnknown:** unrecognized token
- **rtGroup:** "{" or "}"
- **rtfText:** plain text character
- **rtfControl:** token beginning with "\"
- **rtfEOF:** fake class number, indicates end of input stream

Generally, a translator will configure the RTF reader to call particular writer functions when certain kinds of tokens are encountered in the input stream. These functions are known as class callbacks. Writer callbacks can be registered with the reader using RTFSetClassCallback() for each token class.

The reader reads each token, classifies it, and sends it to a token routing function RTFRouteToken(), which tries to find a writer callback function to process the token. Tokens in a given class are ignored if no callback is registered for the class.

Grouping in RTF documents occurs within braces "{" and "}". One kind of group is the destination. The token immediately following the opening brace is a destination control symbol. These indicate such things as headers, footers, footnotes, etc. The reader provides built-in destination readers for the color table, style sheet, and the font table.

The RTF reader also provides a programming interface to enable the writer code to access information about the read token and to act upon it. It is this programming interface that allows the writer code to interact with the reader code to produce a specific RTF translator.

7 rtf2LaTeX 2$_\varepsilon$ Writer Code

At some point, I will try to document the writer source code, but at present it is in too fluid a state. The source code file LaTeX-writer.c is commented though.

8 Acknowledgements

I would not even have attempted this thing had it not been for Paul DuBois' very nicely designed RTF tool. I did not have to bother with parsing the RTF tokens and understanding it. All I had to do was write code to act upon the token. Thanks, Paul, for simplifying it. Another great help has been the DropUnix application framework by Ryan Davis that makes porting between command-line Unix and drag-and-drop Macintosh a matter of changing one line of code. DropUnix itself is based on the drag-and-drop DropShell framework by Leonard Rosenthol, Marshall Clow, and Stephan Somogyi.

Finally, I have to thank Scott Prahl for providing constant feedback and encouragement to keep this going.

9 Legalese