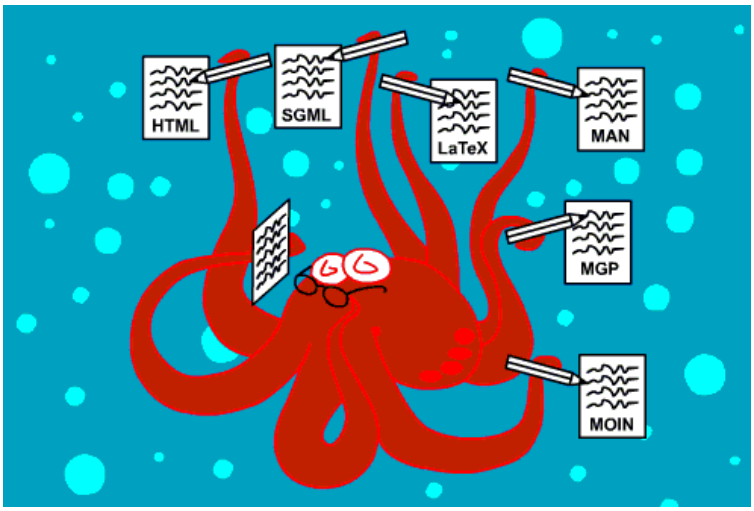
Txt2tags User Guide

http://txt2tags.sf.net



Aurelio Marinho Jargas

v2.3 − May, 2005

Table of Contents

Table of Contents

Part I − Introducing Txt2tags

The First Questions You May Have

This chapter is a txt2tags overview, that will introduce the program purpose and features.

What is it?

Txt2tags is a text formatting and conversion tool.

Txt2tags converts a plain text file with little marks, to any of the supported targets:

- HTML document
- XHTML document
- SGML document
- LaTeX document
- UNIX man page
- Magic Point presentation
- MoinMoin page
- PageMaker 6.0 document
- Plain Text (no marks)

Why should I use it?

You'll find txt2tags really useful if you:

- Need to publish documents on different formats
- Need to maintain updated documents on different formats
- Write technical or manual documents
- Don't know how to write a document on a certain format
- Don't have a specific editor for a certain format
- Want to use a simple text editor to update your documents

And the main motivation is:

- Save time, writing **contents** and forgetting about **formatting**

Why is it a good choice among other tools?

Txt2tags has a very straight way of growing, following basic concepts. These are the highlights:

Source File Readable	Txt2tags marks are very simple, almost natural.
Target Document Readable	As the source file, the target document is also readable, with indentation and short lines.
Consistent Marks	Txt2tags marks are unique enough to fit at all kind of documents and don't be confused with the document contents.
Consistent Rules	As the marks, the rules that applies to them are tied to each other, there are no "exceptions" or "special cases".
Simple Structures	All the supported formatting are **simple**, with no extra−options or complicated behavior modifiers. A mark is just a mark, with no options at all.
Easy to Learn	With simple marks and readable source, the txt2tags learning curve is user friendly.

Nice Examples	The **sample files** included on the package gives real life examples of simple and over−complicated documents written for txt2tags.
Valuable Tools	The **syntax files** included on the package (for vim, emacs, nano and kate editors) help you write documents with no syntax errors.
Three User Interfaces	There is a **Graphical Tk interface** that is very user friendly, a **Web interface** to use it remotely or on the intranet, and a **Command Line interface** for power−users and scripting.
Scripting	With the full featured command line mode, an experienced user can **automatize** tasks and do **post−editing** on the converted files.
Download and Run / Multi−platform	Txt2tags is a single **Python script**. There is no need to compile it or download extra modules. So it runs nicely on *NIX, Linux, Windows and Macintosh machines.
Frequent Updates	The program has an active mailing list with users who suggest corrections and improvements. The author himself is an extensive user at home and at work, so the development won't stop briefly.

Do I have to pay for it?

Absolutely NO!

It's free, GPL, open source, public domain, *<put−your−favorite−buzzword−here>*.

You can copy, use, modify, sell, release as yours. Software politics/copyright is not one of the author's major concerns.

Supported Formatting Structures

The following is a list of all the structures supported by txt2tags.

- header (document title, author name, date)
- section title (numbered or not)
- paragraphs
- font beautifiers
    - bold
    - italic
    - underline
- monospaced font (verbatim)
    - monospaced inside paragraph
    - monospaced line
    - monospaced area (multiline)
- quoted area
- link
    - URL/Internet links
    - e−mail links
    - local links
    - named links
- lists
    - bulleted list
    - numbered list
    - definition list
- horizontal separator line
- image (with smart alignment)
- table (with or without border, smart alignment, column span)

- special mark for raw text (no parsing)
- special macro for current date (with flexible formatting)
- comments (for self notes, TODO, FIXME)

Supported Targets

HTML

Everybody knows what HTML is. (hint: Internet)

Txt2tags generates clean HTML documents, that look pretty and have its source readable. It DOES NOT use javascript, frames or other futile formatting techniques, that aren't required for simple, techie documents. But a separate CSS file can be used if wanted. Txt2tags generates "*HTML 4.0 Transitional*" code.

Since version 2.0, the txt2tags HTML generated code is 100% approved by the w3c validator.

XHTML

It is the new generation of HTML, with more strict rules, as to close all the tags you open. This makes the code easier to parse and understand. For the general purpose, consider it HTML. Txt2tags generates "*XHTML 1.0 Transitional*" code.

Since version 2.0, the txt2tags XHTML generated code is 100% approved by the w3c validator.

SGML

It is a common document format which has powerful sgmltools conversion applications. From a single sgml file you can generate html, pdf, ps, info, latex, lyx, rtf and xml documents. The sgml2* tools also does automatic TOC and break sections into subpages (sgml2html).

Txt2tags generates SGML files in the linuxdoc system type, ready to be converted with sgml2* tools without any extra catalog files or any SGML annoying requirements.

LATEX

The preferred academic document format, it is more powerful than you ever wondered. Full books, complicated formulas and any complex text can be written in LaTeX. But prepare to lost your hair if you will try to write the tags by hand...

Txt2tags generates ready−to−use LaTeX files, doing all the complex escaping tricks and exceptions. The writer just need to worry about the text.

LOUT

Very similar to LaTeX in power, but with an easier syntax using "@" instead "\" and avoiding the need of braces in common situations. Its approach of everything−is−an−object makes the tagging much saner.

Txt2tags generates ready−to−use Lout files, which can be converted do PS or PDF files using the "lout" command.

MAN

UNIX man pages resist over the years. Document formats come and go, and there they are, unbeatable.

There are other tools to generate man documents, but txt2tags has one advantage: one source, multi targets. So the same man page contents can be converted to a HTML page, Magic Point presentation, etc.

MGP

Magic Point is a very handy presentation tool (hint: Microsoft PowerPoint), that uses a tagged language to define all the screens. So you can do complex presentations in vi/emacs/notepad.

Txt2tags generates a ready−to−use .mgp file, defining all the necessary headers for fonts and appearance definitions, as long as ISO−8859 accents support.

HOTSPOT 1: txt2tags created .mgp file uses the XFree86 Type1 fonts! So you do not need to carry TrueType fonts files with your presentation.

HOTSPOT 2: the color definitions for fonts are clean, so even on a poor color palette system (as `startx -- -bpp 8`), the presentation will look pretty!

The key is: convert and use. No quick fixes or requirements needed.

MOIN

You don't know what MoinMoin is? It is a WikiWiki!

Moin syntax is kinda boring when you need to keep `{{{''''adding braces and quotes''''}}}`, so txt2tags comes with the simplified marks and unified solution: one source, multi targets.

PM6

I guess you didn't know, but Adobe PageMaker 6.0 has its own tagged language! Styles, color table, beautifiers, and most of all the PageMaker mouse−clicking features are available on its tagged language also. You just need to access "Import tagged text" menu item. Just for the records, it's an <HTML "like"> tag format.

Txt2tags generates all the tags and already defines a extensive and working header, setting paragraph styles and formatting. This is the hard part. **GOTCHA:** No line breaks! A paragraph must be one single line.

Author's note: *My entire portuguese regular expression's book was written in VI, converted to PageMaker with txt2tags and went to press.*

TXT

TXT is text. The only true formatting type.

Although txt2tags marks are very intuitive and discrete, you can remove them by converting the file to pure TXT.

The titles are underlined, and the text is basically left as is on the source.

Status of Supported Structures by Target

Structure	html	xhtml	sgml	tex	lout	man	mgp	moin	pm6	txt
headers	Y	Y	Y	Y	Y	Y	Y	N	N	Y
section title	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
paragraphs	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
bold	Y	Y	Y	Y	Y	Y	Y	Y	Y	−
italic	Y	Y	Y	Y	Y	Y	Y	Y	Y	−
underline	Y	Y	−	Y	Y	−	Y	Y	Y	−
monospaced font	Y	Y	Y	Y	Y	−	Y	Y	Y	−
verbatim line	Y	Y	Y	Y	Y	Y	Y	Y	Y	−
verbatim area	Y	Y	Y	Y	Y	Y	Y	Y	Y	−
quoted area	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
internet links	Y	Y	Y	−	−	−	−	Y	−	−
e−mail links	Y	Y	Y	−	−	−	−	Y	−	−
local links	Y	Y	Y	N	N	−	−	Y	−	−

named links	Y	Y	Y	−	−	−	−	Y	−	−
bulleted list	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
numbered list	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
definition list	Y	Y	Y	Y	Y	Y	N	N	N	Y
horizontal line	Y	Y	−	Y	Y	−	Y	Y	N	Y
image	Y	Y	Y	Y	Y	−	Y	Y	N	−
table	Y	Y	Y	Y	N	Y	N	Y	N	N
Extras	**html**	**xhtml**	**sgml**	**tex**	**lout**	**man**	**mgp**	**moin**	**pm6**	**txt**
image align	Y	Y	N	N	Y	−	Y	N	N	−
table cell align	Y	Y	Y	Y	N	Y	N	Y	N	N
table column span	Y	Y	N	N	N	N	N	N	N	N

Legend

Y *Supported*

N *Not supported (may be in future releases)*

− *Not supported (can't be done on this target)*

The Three User Interfaces: Gui, Web and Command Line

As different users have different needs and environments, txt2tags is very flexible on how it runs.

There are three User Interfaces for the program, each one with its own purpose and features.

- **Gui**: Written in Tk, brings the windowing and clicking to txt2tags.
- **Web**: Written in PHP, allows users to run txt2tags on the browser, requiring no installation on the client side.
- **Command Line**: Written in Python, it's the program core. All features are available as command line options.

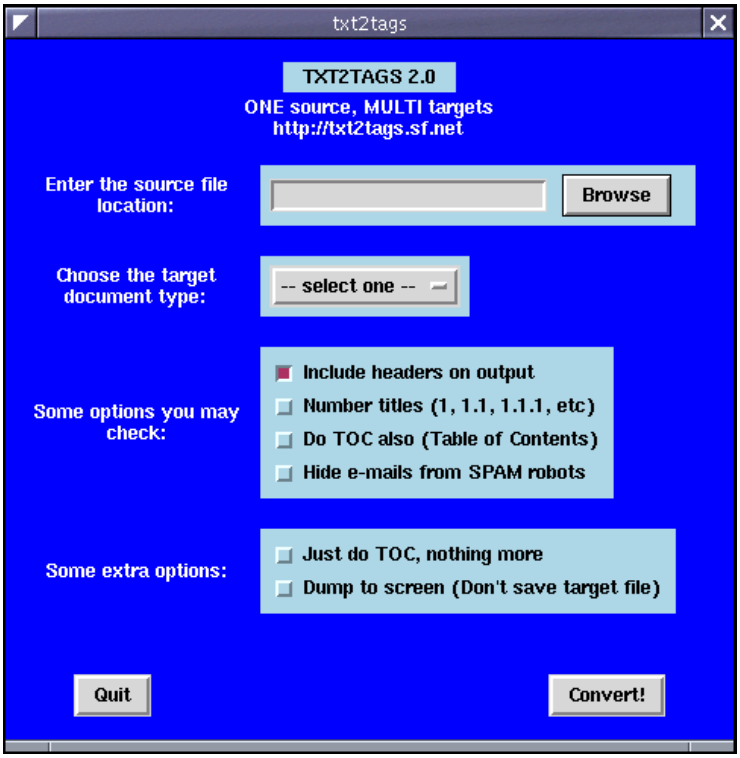
Graphical Tk Interface

Since version 1.0, there is a nice Graphical Interface, that works on Linux, Windows, Mac and others.

The program detects automatically if your system can display the interface, and it is launched when called with no arguments. One can force the Graphical Interface call with the `--gui` option. If some resources are missing, the program will tell.

> **Note:** The Tkinter module is needed. As it comes with the standard Python distribution, you may already have it.

The interface is pretty simple and intuitive:



1. You locate the source .t2t file on the disk and its options are loaded.
2. If the target is still empty, you must choose one.
3. Then there are some options you may choose, but none of them are required.
4. Finally, press the "Convert!" button.

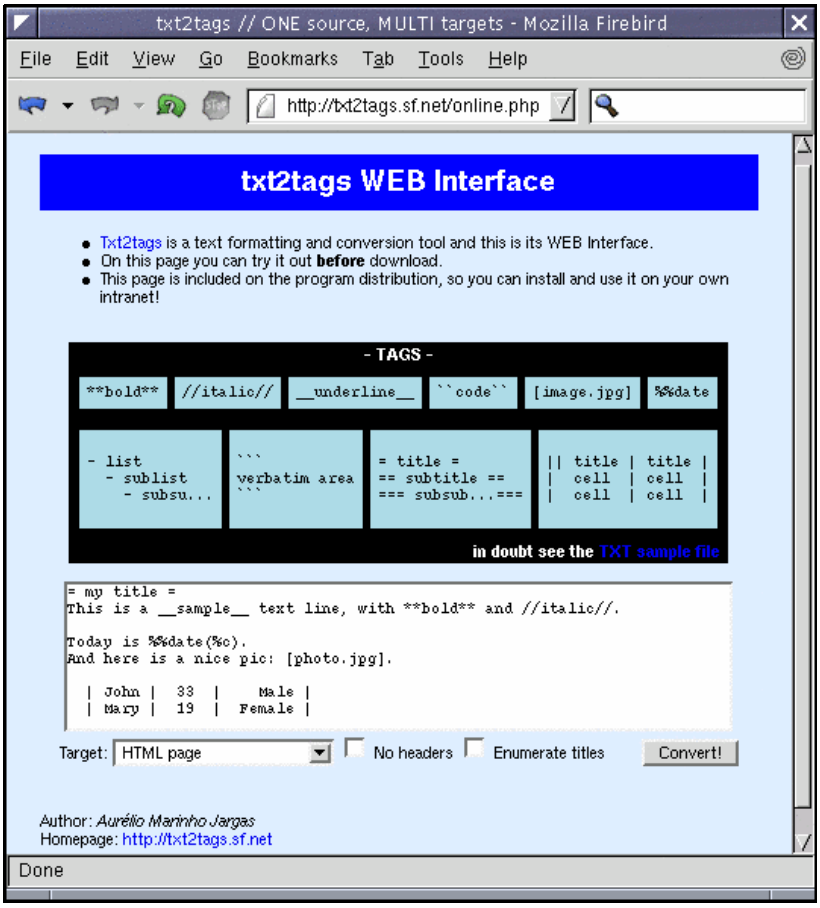
A nice option to check is the "*Dump to screen*", so you can check the resulting code on a separate window, no file is saved at all. When the code is OK, you uncheck it and the file will be saved.

The default interface colors can be changed on the `~/.txt2tagsrc` file, using the `%!guicolors` settings. For example:

```
% set my own colors for the graphical interface (bg1, fg1, bg2, fg2)
%!guicolors: blue white brown yellow
```

Web Interface

The Web Interface is up and running on the Internet at http://txt2tags.sf.net/online.php, so you can use and test the program instantly, before download.



One can also put this interface on the local intranet avoiding to install txt2tags in all machines.

Command Line Interface

For command line power users, the −−help should be enough:

```
Usage: txt2tags [OPTIONS] [infile.t2t ...]

  -t, --target        set target document type. currently supported:
                      html, xhtml, sgml, tex, lout, man, mgp, moin, pm6, txt
  -i, --infile=FILE   set FILE as the input file name ('-' for STDIN)
  -o, --outfile=FILE  set FILE as the output file name ('-' for STDOUT)
  -n, --enum-title    enumerate all title lines as 1, 1.1, 1.1.1, etc
  -H, --no-headers    suppress header, title and footer contents
      --headers       show header, title and footer contents (default ON)
      --encoding      set target file encoding (utf-8, iso-8859-1, etc)
      --style=FILE    use FILE as the document style (like HTML CSS)
      --css-sugar     insert CSS-friendly tags for HTML and XHTML targets
      --css-inside    insert CSS file contents inside HTML/XHTML headers
      --mask-email    hide email from spam robots. x@y.z turns <x (a) y z>
      --toc           add TOC (Table of Contents) to target document
      --toc-only      print document TOC and exit
      --toc-level=N   set maximum TOC level (depth) to N
      --rc            read user config file ~/.txt2tagsrc (default ON)
      --gui           invoke Graphical Tk Interface
  -q, --quiet         quiet mode, suppress all output (except errors)
  -v, --verbose       print informative messages during conversion
  -h, --help          print this help information and exit
  -V, --version       print program version and exit
      --dump-config   print all the config found and exit

  Turn OFF options:
```

```
      --no-outfile, --no-infile, --no-style, --no-encoding, --no-headers
      --no-toc, --no-toc-only, --no-mask-email, --no-enum-title, --no-rc
      --no-css-sugar, --no-css-inside, --no-quiet

  Example:
       txt2tags -t html --toc myfile.t2t

  By default, converted output is saved to 'infile.<target>'.
  Use --outfile to force an output file name.
  If  input file is '-', reads from STDIN.
  If output file is '-', dumps output to STDOUT.
```

Examples

Assuming you have written a `file.t2t` marked file, let's have some converting fun.

Convert to HTML	`$ txt2tags -t html file.t2t`
The same, using redirection	`$ txt2tags -t html -o - file.t2t > file.html`
	.
Including Table Of Contents	`$ txt2tags -t html --toc file.t2t`
And also, numbering titles	`$ txt2tags -t html --toc --enum-title file.t2t`
	.
Contents quick view	`$ txt2tags --toc-only file.t2t`
Maybe enumerate them?	`$ txt2tags --toc-only --enum-title file.t2t`
	.
One liners from STDIN	`$ echo -e "\n**bold**" | txt2tags -t html --no-headers -`
Testing Mask Email feature	`$ echo -e "\njohn.wayne@farwest.com" | txt2tags -t txt --mask-email --no-headers -`
Post-convert editing	`$ txt2tags -t html -o- file.t2t | sed "s/<BODY .*/<BODY BGCOLOR=green>/" > file.html`

Note

Since version 1.6 you can do pre and post processing with the `%!preproc` and `%!postproc` configuration filters.

Part II − OK, I want it. Now what?

Just download the program and run it on your machine.

Download & Install Python

First of all, you must download and install the Python interpreter on your system. If you already have it, just skip this step.

Python is one of the nicest programming languages out there, it works on Windows, Linux, UNIX, Macintosh, and others and it can be downloaded from the Python web site. Installation hints are found on the same site. Txt2tags works with Python version 1.5 or newer.

If you are not sure if you have Python or not, open a console (tty, xterm, MSDOS) and type `python`. If it is not installed, the system will tell you.

Download txt2tags

The official location for txt2tags distribution is on the program site, at http://txt2tags.sf.net/src.

All the program's files are on the tarball (.tgz file), which can be expanded by most of the compression utilities (including Winzip).

Just get the **latest** one (more recent date, higher version number). The previous versions remains for historical purposes only.

Install txt2tags

As a single Python script, txt2tags needs no installation at all.

The only file needed to use the program is the txt2tags script. The other files of the tarball are documentation, tools and sample files.

The fail−proof way to run txt2tags, is calling Python with it:

```
prompt$ python txt2tags
```

If you want to "install" txt2tags on the system as a stand alone program, just copy (or link) the txt2tags script to a System PATH directory and make sure the system knows how to run it.

UNIX/Linux
: Make the script executable (`chmod +x txt2tags`) and copy it to a $PATH directory (`cp txt2tags /usr/local/bin`)

Windows
: Rename the script adding the .py extension (`ren txt2tags txt2tags.py`) and copy it to a system PATH directory (`copy txt2tags.py C:\WINNT`)

After that, you can create an icon on your desktop for it, if you want to use the program's Graphical Interface.

Special Packages for Windows Users

There is also two .EXE distribution files for txt2tags, which install the program on Windows machines with just a few clicks:

- The single txt2tags script for those who already have the Python interpreted installed
- The stand alone version, which doesn't require Python interpreter to run (it has a diet version embedded)

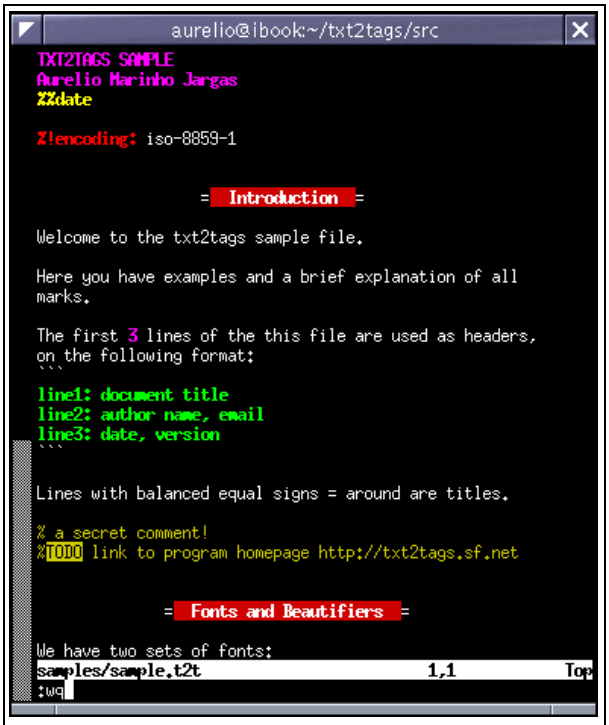
Please visit the *Txt2tags−Win* site to download this packages: http://txt2tags−win.sf.net

Install Text Editor Syntax Highlighting File

Txt2tags comes with handy syntax highlighting files to be used by the following text editors:

- Vim (www.vim.org)
- Emacs (www.emacs.org)
- Nano (www.nano−editor.org)
- Kate (http://kate.kde.org)

This syntax highlighting files have all the txt2tags rules and marks registered, helping the user to write error−free documents. Showing the marks in colors, you see on−the−fly if you wrote it right.



Sample file opened in Vim Editor

Each editor has a different install procedure for a syntax highlighting file, please read the syntax file headers and the editor documentation.

Part III − Writing and Converting Your First Document

Check the Tools

To make the first conversion you will need three things: txt2tags, a text editor and a web browser.

1. Make sure txt2tags is installed and running on your system.
    - **Command Line Interface:** Call "txt2tags" on the command line and the program should give you a "Missing input file" message. If it is not working, try "python /path/to/txt2tags" or even "/path/to/python /path/to/txt2tags" case Python is not on your PATH.
    - **Gui Interface:** Click on the program icon to launch the Gui Interface.
2. Open the text editor your are comfortable with. It can be **any** text editor, from the good old VI to MS Word or OpenOffice.org. Create a brand new empty document to be your first txt2tags one.
3. Launch your favorite web browser to see the results of the conversion to an HTML page.

Write the Document Header

1. Go to the text editor and on the very first line type the document main title: *My First Document*
2. On the second line make a subtitle, inserting this text: *A txt2tags test*
3. Then, on the third line, put some time information, as: *Sunday, 2004*

If everything went right, you should be seeing a three line document with this contents:

```
My First Document
A txt2tags test
Sunday, 2004
```

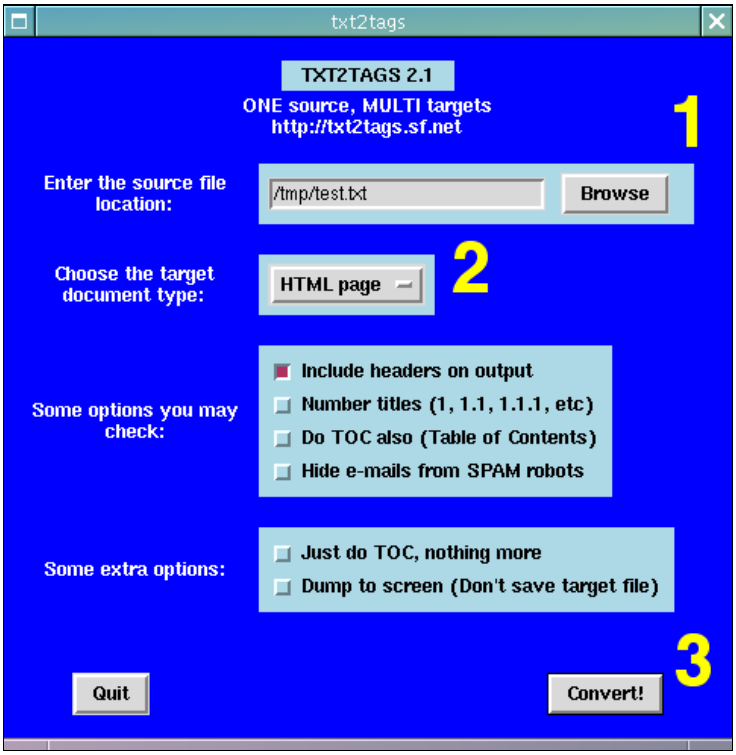
This is just a part of the document, but we can already convert it and check the results.

Now save this document with the name `test.txt`. Pay attention to which folder you are saving the file, you will need to remember it soon.

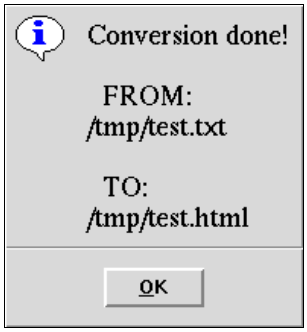
The First Conversion − Gui Interface

If you are in the Command Line Interface, please skip this step and read the next one.

If you are in the Gui Interface, follow this steps:



1. Press the "Browse" button and choose the `test.txt` you just saved (remember the folder!).
2. Back to the first screen, select "HTML page" on the "Target document type" combo.
3. Press the "Convert!" button.



A dialog box will appear, telling you that the file was converted successfully. Note that the generated HTML page was saved on the same folder as the text file, with the "html" extension.

The First Conversion − Command Line Interface

If you are in the Gui Interface, please skip this step and read the next one.

If you are in the Command Line Interface, move to the folder where the file was saved and type this command:

```
txt2tags  --target  html  test.txt
```

Note that there are spaces between the command parts, but no spaces inside the "−−target" string, it is an option. This option is followed by the "html" string, which tells the program to what format your text file will be converted. The last item is the text filename.

If the conversion was done, the results were saved to the `test.html` file and then the program will show you the "*txt2tags wrote test.html*" message. If not, it will tell about some error you may have

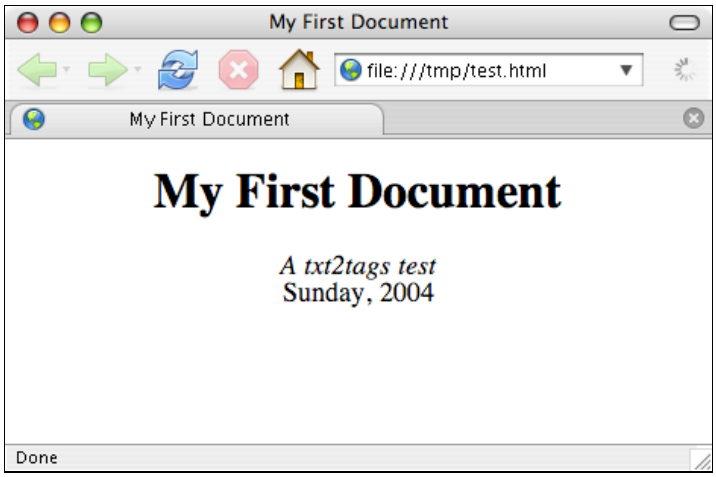
done when typing the command line. Check it with attention.

Here is a sample of how it will be shown on your screen:

```
prompt$ txt2tags −−target html test.txt
txt2tags wrote test.html
prompt$
```

Check the Results

Open the `test.html` file on the Web browser to check if everything is ok.



Here it is! You just typed three simple lines of text and txt2tags made all the work to set the HTML page heading information. Text alignment, sizes, spacing and appearance. See that the main title is placed at the browser title bar also.

You write text, txt2tags does the rest ;)

Tip: You can also use CSS on HTML pages generated by txt2tags, so the page appearance is 100% configurable.

Writing the Document Body

Now back to the text editor, the next step is to type the document contents. You can write plain text as you normally do on email messages. You will see that txt2tags recognizes paragraphs and list of items automatically, you don't have to "mark" them.

Then again: save it, convert and check the results. This is the development cycle of a document in txt2tags. You just focus on the document contents, finishing documents faster than other editors. No mouse clickings, no menus, windows, distraction.

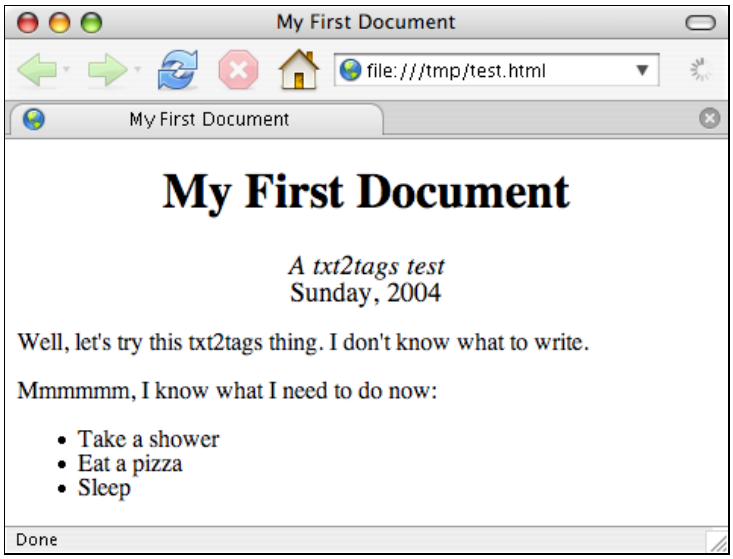
Considering the following contents for the `test.txt` file, which is only plain text, compare the generated HTML page:

```
My First Document
A txt2tags test
Sunday, 2004

Well, let's try this txt2tags thing.
I don't know what to write.

Mmmmmm, I know what I need to do now:
− Take a shower
```

```
- Eat a pizza
- Sleep
```



You can write a full homepage with 0% of HTML knowledge. You don't need to insert any tags. And more, the same text file can be converted to any of the other txt2tags supported formats.

Besides plain text, txt2tags has some very simple marks, that you will use when need some other formating or structures like bold, italic, title, images, table and other. As a quick sample, `**starts for bold**` and `== equals for title ==`. You can learn the marks on the Txt2tags Markup Demo.

Part IV − Mastering Txt2tags Concepts

The .t2t document Areas

Txt2tags marked files are divided in 3 areas. Each area have its own rules and purpose. They are:

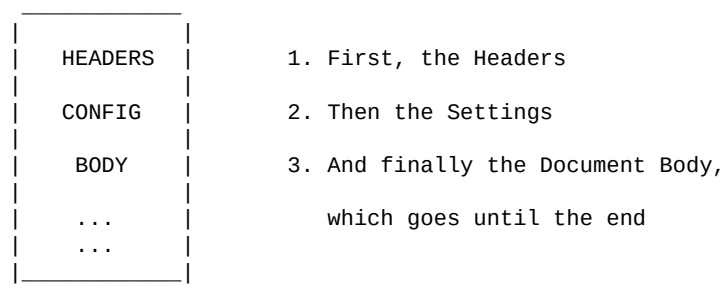
Header Area
: Place for Document Title, Author, Version and Date information. (optional)

Config Area
: Place for general Document Settings and Parser behavior modifiers. (optional)

Body Area
: Place for the Document Content. (required)

As seen above, the first two Areas are optional, being *Body Area* the only required one.

The areas are delimited by special rules, which will be seen in detail on the next chapter. For now, this is a graphical representation of the areas on a document:



In short, this is how the areas are defined:

Headers First 3 lines of the file, or the first line blank for No Headers.

Config Begins right after the Header (4th or 2nd line) and ends when the *Body Area* starts.

Body The first valid text line (not comment or setting) after the *Header Area*.

Full Example

```
My nice doc Title
Mr. John Doe
Last Updated: %%mtime(%c)

%! Target  : html
%! Style   : fancy.css
%! Encoding: iso-8859-1
%! Options : --toc --enum-title

Hi! This is my test document.
Its content will end here.
```

Header Area

Location:

- Fixed position: **First 3 lines** of the file. Period.

- Fixed position: **First line** of the file if it is blank. This means Empty Headers.

The Header Area is the only one that has a fixed position, line oriented. They are located at the first three lines of the source file.

These lines are content−free, with no static information type needed. But the following is recommended for most documents:

- *line 1*: document title
- *line 2*: author name and/or email
- *line 3*: document date and/or version (nice place for `%%date`)

Keep in mind that the first 3 lines of the source document will be the first 3 lines on the target document, separated and with high contrast to the text body (i.e. big letters, bold). If paging is allowed, the headers will be alone and centralized on the first page.

Less (or None) Header lines

Sometimes the user wants to specify less than three lines for headers, giving just the document title and/or date information.

Just let the 2nd and/or the 3rd lines empty (blank) and this position will not be placed at the target document. But keep in mind that even blanks, these lines are still part of the headers, so the document body must start **after** the 3rd line anyway.

The title is the only required header (the first line), but if you leave it blank, you are saying that your document has **no headers**. So the *Body Area* will begin right after, on the 2nd line.

No headers on the document is often useful if you want to specify your own customized headers after converting. The command line option `−−no−headers` is usually required for this kind of operation.

Straight to the point

In short: "Headers are just positions, not contents"

Place one text on the first line, and it will appear on the target's first line. The same for 2nd and 3rd header lines.

Config Area

Location:

- Begins right after the Header Area
  - ♦ Begins on the **4th line** of the file if **Headers** were specified
  - ♦ Begins on the **2nd line** of the file if **No Headers** were specified
- Ends when the Body Area starts
  - ♦ Ends by a non Setting, Blank or Comment line

The Config Area is optional. An average user can write lots of txt2tags files without even know it exists, but the experienced users will enjoy the power and control it provides.

The Config Area is used to store document−specific settings, so you don't have to type them on the command line when converting the document. For example, you can set the default document target type and encoding.

Please read the Settings section for more information about them.

Body Area

Location:

- Begins on the first valid text line of the file
  - ♦ Headers, Settings and Comments are **not** valid text lines
- Ends at the end of the file (EOF)

The body is anything outside Headers and Config Areas.

The body holds the document contents and all formatting and structures txt2tags can recognize. Inside the body you can also put comments for *TODOs* and self notes.

You can use the `--no-headers` command line option to convert only the document body, suppressing the headers. This is useful to set your own headers on a separate file, then join the converted body.

Settings

Settings are special configurations placed at the source document's Config Area that can affect the conversion process. Their syntax is:

%! keyword : value

List of valid keywords:

Keyword	Description
Target	Set the default target to the document be converted to.
Options	Set the default options to be used on the conversion. The format is the same as the command line options.
Style	Set the document style. Used to define a CSS file for HTML/XHTML and to load a package in LaTeX.
Encoding	Set the document Character Set. Used if the document contains accented letters or other not−ASCII characters.
PreProc	Input filter. Sets "find and replace" rules to be applied on the source document.
PostProc	Output filter. Sets "find and replace" rules to be applied on the converted document.

Example:

```
%! Target  : html
%! Options : --toc --toc-level 3
%! Style   : fancy.css
%! Encoding: iso-8859-1
%! PreProc : "AMJ"        "Aurelio Marinho Jargas"
%! PostProc: '<BODY.*?>'  '<BODY bgcolor="yellow">'
```

Command Line Options

The fastest way of changing the txt2tags default behavior is to use command line options. This options are available on the Command Line Interface only, not on Gui or Web.

Just like the other system's tools, the program do accept a set of predefined options. An option is an hyphen followed by a letter or two hyphens followed by one or more words, like `−t` and `−−target`. Talking about the target option, it is the only required one, the others are optional.

Options that are generally used are `−−outfile` to define a customized output file name, `−−toc` to turn on the automatic TOC generation and `−−encoding` to set the document character set. Most of the options can be turned off prefixing a "no−" before its name, for example: `−−no−encoding` and `−−no−toc`.

You can register the desired options for a source file inside its Config Area, using the `%!options` setting. This way you don't have to type them on the command line anymore. Example: `%!options: −−toc −o mydoc.html`. The exception is the target specification, that has its own setting: `%!target: html`.

Do use the `−−help` option to get a complete list of all the options available in txt2tags.

User Configuration File (RC File)

The user configuration file (also called RC file) is a central place to store the settings that will be shared by ALL converted files. If you keep inserting the same settings on every .t2t file you write, move it to the RC file and it will be used globally, for existing and future source files.

The default location of this file depends on your system. It can also be specified by the user, using an environment variable.

	RC file location
Windows:	`%HOMEPATH%\_t2trc`
Linux and other:	`$HOME/.txt2tagsrc`
User defined:	`T2TCONFIG` env var

The format of the settings is exactly the same as the ones used on the .t2t files Config Area. There is a sample RC file on the tarball on `doc/txt2tagsrc`. Example:

```
% my configs

%%% Always use CSS−friendly tags in HTML
%!options(html): −−css−suggar

%%% Change the default TOC depth for all targets
%!options: −−toc−level 4

%%% Set the default encoding for all documents
%!options: −−encoding iso−8859−1
```

Any line that is not blank, a comment or a valid config line will raise error when txt2tags runs. So be careful when editing this file.

Txt2tags automatically apply the RC file contents into any source file it is converting. If you want to disable this behavior for a specific file, do use the `−−no−rc` command line option.

Configuration Loading Order and Precedence

There are three ways of telling txt2tags which options and settings to use, and this is the order that they are read and applied:

1. The user configuration file (RC) settings
2. The source document Config Area settings
3. The command line options

First txt2tags reads the RC file contents (if any) and apply its configurations on the current source file. Then it scans the source document Config Area for settings and if found, they are applied also, overriding the RC ones in case of conflict. Finally comes the command line options, stronger than the other two.

So, if the document encoding was defined on the three resources, the command line will be the one used.

%!include command

The `include` command is used to paste the contents of an external file into the source document body. It is not a config, but a command, and it is valid on the document Body Area.

The `include` command is useful to split a large document into smaller pieces (like chapters in a book) or to include the full contents of an external file into the document source. Sample:

```
My first book
Dr. John Doe
1st Edition

%!include: intro.t2t
%!include: chapter1.t2t
%!include: chapter2.t2t
...
%!include: chapter9.t2t
%!include: ending.t2t
```

You just inform the filename after the `%!include` string. The optional target specification is also supported, so this is valid either:

```
%!include(html): file.t2t
```

Note that include will insert the file Body Area into the source document. The included file Header and Config Areas are ignored. This way you can convert the included file alone or inside the main document.

But there's another three types of include:

- Verbatim include
- Raw include
- Tagged include

The **Verbatim** type includes a text file preserving its original spaces and formatting, just like if the text was inside the txt2tags VERB area (```). To specify this type, enclose the filename with backquotes:

```
%!include: ``/etc/fstab``
```

The **Raw** type includes a text file as is, not trying to find and parse txt2tags marks on it, just like if the text was inside the RAW area ("""). To specify this type, enclose the filename with double quotes:

```
%!include: ""nice_text.txt""
```

And the **Tagged** type is passed directly to the resulting document, with NO parsing or escaping performed by txt2tags. This way you can include additional tagged parts to your document. Useful for

default header or footer information, or more complicated tagged code, unsupported by txt2tags:

```
%!include(html): ''footer.html''
```

Note that the filename is enclosed with single quotes, and as the text inserted is already parsed, you must specify the target to avoid mistakes.

%!includeconf command

The `includeconf` command is used to include configurations from an external file into the current one. This command is valid inside the source document Config Area only.

It is useful to share the same config for multiple files, so you can centralize it. On any file do you want to include that central configuration, put a `includeconf` call. Example:

```
My First Document
John Doe
July, 2004

%!includeconf: config.t2t

Hi, this is my first document.
```

The format inside the included file is the same as in the RC file.

Part V − Mastering Marks

Overview of all txt2tags marks:

Basic	**. . . . . .**	**Beautifiers**	**. . . . . .**
Headers	First 3 lines	*Bold*	**words**
Title	= words =	*Italic*	//words//
Numbered title	+ words +	*Underline*	__words__
Paragraph	words	*Monospaced*	``words``
Text Blocks	**. . . . . .**	**Other**	**. . . . . .**
Quote	<TAB>words	*Separator line*	−−−−−−−−−−−...
List	− words	*Strong line*	==========...
Numbered list	+ words	*Links*	[label url]
Definition list	: words	*Image*	[filename.jpg]
Verbatim line	``` words	*Comment*	% comments
Verbatim area	```\n lines \n```	*Raw text*	""words""
Raw line	""" words	*Table*	| cell1 | cell2 | cell3...
Raw area	"""\n lines \n"""	*Anchor*	= title =[anchor]

General Rules:

- **Headers** are the first three document lines, marks are not interpreted.
- **Titles** are balanced "=" or "+" chars around the title text. The more chars, more deep is the title.
- **Beautifiers** don't accept spaces between the marks and its contents.
- The **Comment** mark "%" must be at the line beginning (first column).
- **Images** filename must end in GIF, JPG, PNG or similar.
- The only **multiline** marks are the Verbatim and Raw areas.
- No mark is **interpreted** inside Verbatim or Raw.
- The **Separator/Strong lines** must have at least 20 chars.
- Quote and lists **(un)nesting** is defined by indent.
- A **Table title** line is defined by two || at the beginning of the line.

Headers

- **Description:** Identifies the document headers
- **Properties:** Multiline, FreeSpaces, !Align, !Nesting
- **Contains:** Macros
- **Syntax:**
  - The first 3 lines of the source file.
  - Leave the first line blank to not specify headers at all. Nice for command line oneliners or customized headers.
  - Leave the second and/or third lines blank to omit parts of header.
- **Details:**
  - Marks are NOT interpreted
  - The first 3 lines will be the first 3 lines on the target document, with high contrast to text body, or will be placed alone on the first page (if paging is allowed).
  - The headers are content−free, with no static information type needed. But the following is recommended for the most documents:

◊ Line 1: Document title
◊ Line 2: Author name and/or email
◊ Line 3: Document date and/or version (nice place for %%mtime)

Title, Numbered Title

- **Description:** Identifies a (numbered or not) section title
- **Properties:** !Multiline, FreeSpaces, !Align, !Nesting
- **Contains:** Raw
- **Syntax:**
  - ♦ For Numbered Title, just change "=" by "+" on the following rules
  - ♦ Balanced equal signs around, `=like this=`
  - ♦ More signs, more sublevels: `=title=`, `==subtitle==`, `===subsubtitle===`, ...
  - ♦ There is a maximum of 5 levels, `=====like this=====`
  - ♦ Unbalanced equals are not title, `=like this===`
  - ♦ Free spacing inside the marks are allowed, `= like this =`
  - ♦ Titles can have an anchor `=like this=[anchor]`. To link to an anchor create a `[local link #anchor]`
- **Details:**
  - ♦ Marks are NOT interpreted
  - ♦ Macros are NOT interpreted

Paragraph

- **Description:** Identifies a paragraph of text
- **Properties:** Multiline, FreeSpaces, !Align, !Nesting
- **Contains:** Macros, Beautifiers, Raw, Links, Image, Comment
- **Syntax:**
  - ♦ Paragraphs are groups of lines delimited by blank lines
  - ♦ Other blocks like lists, quote, table or verbatim also ends a paragraph

Comment

- **Description:** Used to insert text that will not appear on the target document
- **Properties:** !Multiline, !<FreeSpaces, !Align, !Nesting
- **Contains:** −
- **Syntax:**
  - ♦ A line beginning with a percent char at the first column, `% like this`
  - ♦ NO leading spaces
- **Details:**
  - ♦ As comments, they're not showed on the converted text
  - ♦ Not a block, so each comment line must begin with %
  - ♦ Useful for TODO and FIXME reminders and editor's notes

Bold, Italic, Underline

- **Description:** Used to insert a bold/italic/underline text inside a paragraph, table, list or quote
- **Properties:** !Multiline, !FreeSpaces, !Align, Nesting
- **Contains:** Macros, Beautifiers, Raw, Links, Image
- **Syntax:**
  - ♦ Two starts around for bold, `**like this**`
  - ♦ Two slashes around for italic, `//like this//`

    - Two underlines around for underline, `__like this__`
    - The marks must be glued with the contents (no spaces): `** this ** is invalid`
- **Details:**
    - All the beautified text must be on a single line of the source file, no line breaks inside
    - Macros are allowed inside beautifiers: `**%%date**`
    - You can mix beautifiers one inside another, `**__like__ //this//**`

Monospaced

- **Description:** Used to insert a monospaced text inside a paragraph, table, list or quote
- **Properties:** !Multiline, !FreeSpaces, !Align, !Nesting
- **Contains:** −
- **Syntax:**
    - Two backquotes around, ``` ``like this`` ```
    - The marks must be glued with the contents (no spaces): ``` `` this `` is invalid ```
- **Details:**
    - Marks are NOT interpreted
    - Macros are NOT interpreted
    - All the monospaced text must be on a single line of the source file, no line breaks inside
    - In some targets, the internal spacing is maintained, in others the consecutive spaces are squeezed to one
    - You can make a bold monospaced text enclosing it inside bold marks: `**``monobold``**`. The same applies to the other beautifiers as `//``italic``//` and `__``underline``__`.

Verbatim Line, Verbatim Area

- **Description:** Used to insert programming codes or other pre−formatted text, preserving spacing and line breaks, and using a monospaced font
- **Properties:** Multiline, !FreeSpaces, !Align, !Nesting
- **Contains:** −
- **Syntax: Verbatim Line:**
    - A line beginning with 3 consecutive backquotes, followed by a space, followed by the text, ``` ``` like this ```
    - The backquotes must be at the start of the line, no spaces before
- **Syntax: Verbatim Area:**
    - A line with exactly 3 consecutive backquotes ```` ``` ````, followed by text lines, followed by another line with exactly 3 consecutive backquotes ```` ``` ````
    - NO spaces allowed before or after the marks
- **Details:**
    - Marks are NOT interpreted
    - Macros are NOT interpreted
    - If the end of the source file (EOF) is hit, the opened Verbatim Area is closed

Separator Line, Strong Line

- **Description:** Identifies a separator or strong line
- **Properties:** !Multiline, FreeSpaces, !Align, !Nesting
- **Contains:** −
- **Syntax:**
    - The separator line can be composed by dashes "−" or underscores "_"
    - The strong line is composed by equals "="

- Use at least least 20 dashes/underscores/equal signs
- Optional spaces can be placed at the line start or end
- Any other characters on the line invalidate the mark

- **Details:**
  - If the target does not have separator line support, a commented line is used instead
  - The strong line may have different behaviors on some targets:
    - A larger separator line
    - A pause on presentation formats, like Magic Point
    - A page break in paged targets, like LaTeX

Links, Named Links

- **Description:** Identifies a remote (Internet) or local link
- **Properties:** !Multiline, !FreeSpaces, !Align, !Nesting
- **Contains:** Macros, Raw, Image
- **Syntax:**
  - Any valid internet URL, ftp, news or email address is detected and converted automatically
  - The protocol (http, https, ftp) is optional, `www.likethis.com`
  - A name can be used for a link: `[click here www.url.com]`
  - An image can point to a link: `[[image.jpg] www.url.com]`
  - Macros are allowed on the link address: `[see source %%infile]`
  - Macros are allowed on the link name: `[mirror of %%outfile www.url.com]`
  - All the link specification must be on a single line of the source file, no line breaks inside
- **Details:**
  - If the target does not have link support, they're just underlined

Quote

- **Description:** Identifies a quoted (indented) line
- **Properties:** Multiline, !FreeSpaces, !Align, Nesting
- **Contains:** Macros, Beautifiers, Quote, Raw, Bars, Links, Image, Comment
- **Syntax:**
  - A line that starts with a tabulation (TAB) character
  - More TABs at the start increase the quote depth
  - Lists and tables are not allowed inside quote
- **Details:**
  - If the end of the source file (EOF) is hit, the opened Quote is closed
  - Some targets may not support quote nesting, then the subquotes lines are moved up to the mother quote level.
  - There is not a limit for subquotes depth. But some targets may have restrictions, so the subquotes than are deeper than the maximum level are moved up.

List, Numbered List, Definition List

- **Description:** Identifies the start of a list item
- **Properties:** Multiline, !FreeSpaces, !Align, Nesting
- **Contains:** Macros, Beautifiers, Lists, Table, Verbatim, Raw, Bars, Links, Image, Comment
- **Syntax:**
  - A line that starts with a dash/plus/colon followed by exactly one space
  - The first list char can NOT be a space (exception: definition lists)
  - Optional spaces (regular spaces, not TAB) at the line beginning define sublists depth

(nesting)
    - Sublists end with a less depth item (from parent list) or with an empty item
    - All opened lists are closed with two consecutive blank lines
- **Details:**
    - If the end of the source file (EOF) is hit, all opened lists are closed
    - Lists can be mixed, like a definition list inside a numbered list.
    - Some targets may not support list nesting, then the sublists items are moved up to the mother list level.
    - There is not a limit for sublists depth. But some targets may have restrictions, so the sublists than are deeper than the maximum level are moved up.

Image

- **Description:** Identifies an image
- **Properties:** !Multiline, !FreeSpaces, Align, !Nesting
- **Contains:** Macros
- **Syntax:**
    - An image filename enclosed between brackets, `[likethis.jpg]`
    - The filename must end in an image extension like PNG, JPG, GIF, ... (case doesn't matter)
    - Symbols are allowed on the filename, `[likethis!~1.jpg]`
    - Macros are allowed on the filename, `[report−%%date(%Y−%m−%d).png]`
    - NO spaces allowed on the filename, `[like this.jpg]`
    - NO spaces allowed on the brackets, `[ likethis.jpg ]`
- **Details:**
    - If the target does not have image support, the image filename is shown inside (parenthesis).
    - The position of the mark on the line defines the image alignment:
        - ◊ `[LEFT.jpg]` blablablabla
        - ◊ blablablabla `[CENTER.jpg]` blablablabla
        - ◊ blablablabla `[RIGHT.jpg]`

Table

- **Description:** Delimits a table row, with any number of columns
- **Properties:** Multiline, FreeSpaces, Align, !Nesting
- **Contains:** Macros, Beautifiers, Raw, Links, Image, Comment
- **Syntax:**
    - A leading pipe "|" identifies a table row
    - A leading double pipe "||" identifies a table title row
    - Leading spaces before first pipe identifies table centered align
    - The fields are separated by the " | " string (space pipe space)
    - A final pipe "|" at the first table row sets visible borders
    - A final pipe "|" at the other table rows are ignored (just cosmetic)
    - Closing a cell with more than one pipe "|" identifies column span: "||" for 2 columns, "|||" for 3 and so on
    - Natural spaces inside each cell identifies its alignment
    - Example: `| table | row | with | five | columns |`
- **Details:**
    - All the table row data must be on a single line of the source file, no line breaks inside
    - Targets with column−oriented align (like sgml and LaTeX), uses the first table row align as the default for the other rows
    - Any non−table line closes the opened table, except comment lines
    - The cell count is flexible, each table row can have a different number of cells
    - Currently there's no way to specify row span

- If the target does not have table support, the table lines are considered a Verbatim Area

Raw, Raw Line, Raw Area

- **Description:** Used to "protect" some text from parsing, so marks and macros inside it will not be expanded.
- **Properties:** !Multiline, !FreeSpaces, !Align, !Nesting
- **Contains:** −
- **Syntax: Raw:**
  - Two double quotes around, `""like this""`
  - Marks glued with the contents (no spaces)
- **Syntax: Raw Line:**
  - A line beginning with 3 consecutive double quotes, `""" like this`
  - The double quotes must be at the start of the line, no spaces before
  - Use a space after the double quotes to separate them from the text
- **Syntax: Raw Area:**
  - A line with exactly 3 consecutive double quotes, followed by text lines, followed by another line with exactly 3 consecutive double quotes
  - NO spaces allowed before or after the marks
- **Details:**
  - Marks are NOT interpreted
  - Macros are NOT interpreted
  - If the end of the source file (EOF) is hit, the opened Raw Area is closed

Part VI − Mastering Macros

Macros are special purpose keywords, that are expanded on conversion time. They are used to insert dynamic information, for example the current date or information about the document source.

A macro is represented by the %% chars followed by its name, as `%%date`. Some macros do accept an optional formatting string inside parenthesis, right after the macro name, as `%%date(%Y−%m−%d)`. This format string mixes common text with special directives, identified by a percent sign % followed by an identification character. If not format string is given, the default format is used.

Macro Name	Expands to...	Default Format
%%date	The current date	%Y%m%d
%%mtime	The source file modification time	%Y%m%d
%%infile	The source file path	%f
%%outfile	The output file path	%f
%%toc	The document TOC (Table of Contents)	−

General rules:

- The macro name is case−insensitive, so `%%date`, `%%DaTe` and `%%DATE` are identical
- Macros are valid at the document Header and Body Areas, except %%toc that is valid on Body Area only
- A macro starts the Body Area if it is found on the Config Area
- A macro can be placed at any part of the line, various per line (except %%toc, valid when alone in a line)
- A macro can be used inside links and images marks (except %%toc)
- Macros are not expanded in Titles, Verbatim and Raw Areas

Full example (bold text are expanded macros):

> This is the Txt2tags User Guide, converted to **html** by txt2tags from the **−** source file. The conversion was done at **2005−06−14 23:16:09**, but the last change on the source document was made on **2005−06−14 23:16:09**. Both source and converted file reside on the directory.

%%date

The `%%date` macro expands to the current date and time. It is very useful on the document headers or footer, to register the date when the document was generated. To expand to the source document last modification time, see the %%mtime macro.

This macros accepts several formatting directives. The full list can be found in the Python site. Here are the most commonly used:

Directive	Description
%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%c	Locale's appropriate date and time representation.

%d	Day of the month as a decimal number [01,31].
%H	Hour (24−hour clock) as a decimal number [00,23].
%I	Hour (12−hour clock) as a decimal number [01,12].
%m	Month as a decimal number [01,12].
%M	Minute as a decimal number [00,59].
%p	Locale's equivalent of either AM or PM.
%S	Second as a decimal number [00,61]. (1)
%x	Locale's appropriate date representation.
%X	Locale's appropriate time representation.
%y	Year without century as a decimal number [00,99].
%Y	Year with century as a decimal number.
%%	A literal "%" character.

Examples:

Macro	−−>	Results for on 2005, Jun 14 at 23:16
%%date(Converted on: %c)	−−>	Converted on: Tue Jun 14 23:16:09 2005
%%date(%Y−%m−%d)	−−>	2005−06−14
%%date(%I:%M %p)	−−>	11:16 PM
%%date(Today is %A, on %B.)	−−>	Today is Tuesday, on June.

%%mtime

The `%%mtime` macro expands to last modification time of the source document. It is useful to register when the file was last changed. This macro is a "sister" of the %%date macro, so it accepts exactly the same formatting directives.

As an example, this User Guide source file was last edited on **Tue Jun 14 23:16:09 2005**. This date was expanded from `%%mtime(%c)`.

%%infile

The `%%infile` macro expands to the source file location on the system. It is useful to make those "*see the source of this file*" links on HTML pages. Provide such link is a friendly attitude with beginners, so they can use your source as a sample for their own page.

This macro accept the following formatting directives:

%<char>	Description	Output for this User Guide source
%f	File name	−
%F	File name (without extension)	−
%e	File extension	
%p	Absolute file path	−
%d	File path (directories only)	

%D	File path (parent dir only)	
%%	Literal percent char	%

Examples:

Source	−−>	Expanded
This Guide parent dir is %%infile(%D).	−−>	This Guide parent dir is .
I do use the %%infile(%e) file extension.	−−>	I do use the file extension.
[See the source %%infile]	−−>	See the source
Converted to XHTML, I'll be %%infile(%F).xhtml	−−>	Converted to XHTML, I'll be −.xhtml

Note: The macro is expanded to "−" if the source file is STDIN.

%%outfile

The `%%outfile` macro expands to the converted file location on the system. It is useful to its name inside the document Body or Headers. This macro is a sister of the %%infile macro and do accept exactly the same formatting directives.

Examples:

Source	−−>	Expanded
You are reading the %%outfile file.	−−>	You are reading the userguide−pdf.html file.
txt2tags −t %%outfile(%e) −i %%infile −o %%outfile	−−>	txt2tags −t html −i − −o userguide−pdf.html

Note: The macro is expanded to "−" if the output file is STDOUT.

%%toc

The `%%toc` macro expands to the document's Table of Contents. It is useful for you to specify exactly where you want the TOC to be placed. You can even use the macro more than one time and place the TOC at the end of the document also, for example. This Guide is using %%toc to position the TOC.

Different from the other macros, this one does not accept a format string and has its own special rules:

- Valid at the document Body Area only
- Must be alone on the line (leading and trailing spaces are allowed)
- Must be used together with −−toc command line option, or it will be ignored
- The default automatic TOC positioning/formatting is disabled when a %%toc is found

Part VII − Mastering Settings

Settings are special configurations placed at the source document's Config Area that can affect the conversion process. The Settings are all optional. The average user can live fine without them. But they are addictive, if you start using them, you'll never stop :)

Setting lines are *special comment lines*, marked by a leading identifier ("!") that makes them different from plain comments. The syntax is just as simple as variable setting, composed by a keyword and a value, separated from each by a colon (":").

%! keyword : value

Syntax details:

- The exclamation mark must be placed together with the comment char ("%!"), no spaces between them.
- The spaces around the keyword and the separator are optional.
- Both keyword and value are case insensitive (case doesn't matter).

Rules:

- Settings are valid only inside the Config Area, and are considered plain comments if found on the document Body.
- If the same keyword appears more than one time on the Config Area, the last found will be the one used. Exception: options, preproc and postproc, which are cumulative.
- A setting line with an invalid keyword will be considered a plain comment line.
- This settings have precedence over RC file, but not on command line options.

%!Target

Using the target setting, a default target format is defined for the document:

```
%!target: html
```

This way the user can just call

```
$ txt2tags file.t2t
```

And the conversion will be done, to the specified target.

The target setting does not support optional target specification. That doesn't make sense, as `%!target(tex): html`.

%!Options

Writing long command lines every time you need to convert a document is boring and error prone. The Options setting let the user save all the converting options together with the source document. This also ensures that the document will always be converted the same way, with the same options.

Just write it with no syntax errors, as you were on the real command line. But omit the "txt2tags" program call on the beginning, the target specification and the source filename from the ending.

For example, if you do use this command line to convert your document:

```
$ txt2tags −t html −−toc −−toc−level 2 −−enum−title file.t2t
```

You can save yourself from typing pain using this Options setting inside the document source:

```
%!target: html
%!options(html): −−toc −−toc−level 2 −−enum−title
```

As the real command line is now just "`txt2tags file.t2t`", you can run the conversion right inside your favorite text editor, while editing the document source. In Vi, this is:

```
:!txt2tags %
```

%!Encoding

The Encoding setting is needed by non−english writers, who uses accented letters and other locale specific details, so the target document *Character Set* must be customized (if allowed).

The valid values for the Encoding setting are the same charset names valid for HTML documents, like *iso−8859−1* and *koi8−r*. If you're not sure which encoding you should use, this complete (and long!) list should help.

The LateX target use alias names for encoding. This is not a problem for the user, because txt2tags translate the names internally. Some examples:

txt2tags/HTML	>	LaTeX
windows−1250	>>>	cp1250
windows−1252	>>>	cp1252
ibm850	>>>	cp850
ibm852	>>>	cp852
iso−8859−1	>>>	latin1
iso−8859−2	>>>	latin2
koi8−r	>>>	koi8−r

If the value is unknown to txt2tags, it will be passed "as is", allowing user to specify custom encodings.

%!PreProc

The PreProc is an input filter used on the source document. It is a "find and replace" feature, applied right after the line is read from the document source, before any parsing by txt2tags.

It is useful to define some abbreviations for common typed text, as:

```
%!preproc JJS          "John J. Smith"
%!preproc RELEASE_DATE "2003−05−01"
%!preproc BULLET       "[images/tiny/bullet_blue.png]"
```

So the user can write a line like:

```
Hi, I'm JJS. Today is RELEASE_DATE.
```

And txt2tags will "see" this line as:

```
Hi, I'm John J. Smith. Today is 2003−05−01.
```

This filter is a component that acts between the document author and the txt2tags conversion. It's like a first conversion before the "real" one. Its behavior is just like an external Sed/Perl filter, called this way:

```
$ cat file.t2t | preproc−script.sh | txt2tags −
```

So the txt2tags parsing will begin after all the PreProc substitutions were applied.

%!PostProc

The PostProc is an output filter used on the converted document. It is a "find and replace" feature, applied after all txt2tags parsing and processing is done.

It is useful to do some refinements on the generated document, change tags and add extra text or tags. Quick samples:

```
%!postproc(html): '<BODY.*?>' '<BODY BGCOLOR="green">'
%!postproc(tex) : "\\newpage" ""
```

These filters change the background color of the HTML page and remove the page breaks on the LaTeX target.

The PostProc rules are just like an external Sed/Perl filter, called this way:

```
$ txt2tags −t html −o− file.t2t | postproc−script.sh > file.html
```

Before this feature was introduced, it was very common to have little scripts to "adjust" the txt2tags results. These scripts were in fact just lots of sed (or alike) commands, to do "substitute this for that" actions. Now this replacement strings can be saved together with the document text, and the plus is to use the Python powerful Regular Expression machine to find patterns.

%!Style

- Useful in HTML and XHTML targets, it defines a CSS file for the target document.
- Useful in LaTeX target, to load `\usepackage` modules.
- The same effect is achieved with the command line option `−−style`.
- The −−style option is stronger than %!style. If both are used, −−style wins.

Defining a Setting for a Specific Target

All the settings (except %!target) can be glued with a specific target using the `%!key(target): value` syntax. This way user can define different config for different targets.

This is specially useful in the pre/postproc filters, but is applicable to all settings. For example, defining different styles for HTML and LaTeX:

```
%!style(html): fancy.css
%!style(tex) : amssymb
```

For the options setting it's very useful to adjust the converted document:

```
%!target: sgml
%!options(sgml): −−toc
%!options(html): −−style foo.css
%!options(txt ): −−toc−only −−toc−level 2
```

In this example, the default target is Sgml and it will use TOC. If the user run `txt2tags −t html file.t2t`, only the HTML options will be used, so the converted file will use "foo.css" style file and will have no TOC.

Details for PreProc and PostProc Filters

- Filters are a "find and replace" feature (think SED)
- Filters do not follow the "last found, one used" schema, they're cumulative. You can define as many filters as needed, with no limit. They will be applied on the same order as defined.
- Different from other settings, both the target specific filters and the generic ones (all targets) are used. On the following example, both filters are used on the HTML target:

```
%!postproc      :   this   that
%!postproc(html):   that   other
```

- The filters must receive exactly TWO arguments
- Special escapes as `\n` (line break) and `\t` (tabulation) are interpreted
- To delete some text, change it by an empty string

```
%!postproc: "undesired string" ""
```

- To avoid problems, always use the explicit target form when using PostProc to change tags: `%!PostProc(target): <this> <that>`
- PREproc is applied right after the line is read, and POSTproc is applied after all the parsing was made. They are exactly the same as (UUOC ahead):

```
$ cat file.t2t | preproc.sh | txt2tags | postproc.sh
```

- The first part of a filter (the "search for" part) is not read as a regular string, but a Regular Expression pattern. If you don't know what these expressions do, don't worry, you may never have to. Just keep in mind that you will need to "escape" some characters to use them. To escape is to prefix the character with a backslash "\". Here is the list:

```
\*  \+  \.  \^  \$  \?  \(  \)  \{  \[  \|  \\
```

- Python Regular Expressions are available! They're similar to Perl Regexes (PCRE). Example: Change all opening and closing "B" tags to "STRONG" on HTML:

```
%!postproc(html):   '(</?)B>'   '\1STRONG>'
```

- The filter arguments can be passed on 3 ways:
    1. A single unquoted word as FOO (no spaces)
    2. A string double quoted as "FOO"
    3. A string single quoted as 'FOO'
- If your pattern has double quotes, protect it with single quotes and vice−versa. Some valid samples:

```
%!postproc:   PATT    REPLACEMENT
%!postproc:  "PATT"  "REPLACEMENT"
%!postproc:  'PATT'  'REPLACEMENT'
%!postproc:   PATT   "REPLACEMENT"
%!postproc:  "PATT"  'REPLACEMENT'
```

Part VIII − Black Magic

This chapter is really not recommended for newbies. It demonstrates how to do strange things with txt2tags filters, abusing of complex patterns and Regular Expressions.

> **BEWARE!** The following procedures are NOT encouraged and can break things. Even some text from the document source can be lost on the conversion process, not appearing on the target document. Just use these tactics if you really need them and know what you are doing.

Filters are a powerful feature, but can be dangerous!

Bad filters do generate unexpected results.

Keep that in mind, please.

Inserting Multiple Lines with %!PostProc (like CSS rules)

In filters, the replacement pattern can include multiple lines using the `\n` line break char.

This can be handy for including really short CSS rules on HTML target, with no need to create a separate file:

```
%!postproc: <HEAD>      '<HEAD>\n<STYLE TYPE="text/css">\n</STYLE>'
%!postproc: (</STYLE>)  'body     { margin:3em               ;} \n\1'
%!postproc: (</STYLE>)  'a        { text-decoration:none     ;} \n\1'
%!postproc: (</STYLE>)  'pre,code { background-color:#ffffcc ;} \n\1'
%!postproc: (</STYLE>)  'th       { background-color:yellow  ;} \n\1'
```

All the filters are tied to the first one, by replacing a string that it has inserted. So a single "<HEAD>" turns to:

```
<HEAD>
<STYLE TYPE="text/css">
body     { margin:3em               ;}
a        { text-decoration:none     ;}
pre,code { background-color:#ffffcc ;}
th       { background-color:yellow  ;}
</STYLE>
```

Creating "Target-Specific" Contents with %!PreProc

Sometimes you need to insert some text on a specific target, but not on the others. This kind of strange behavior can be done using some PreProc tricks.

The idea is to insert this extra text on the document source as comments, but mark it in a way that a target-specific filter will "uncomment" those lines.

For example, if an extra paragraph must be added only in HTML target. Place the text as special comments, like this:

```
%html% This HTML page is Powered by [txt2tags http://txt2tags.sf.net].
%html% See the source TXT file [here source.t2t].
```

As those lines start with %, they are plain comments lines and will be ignored. But when adding this

special filter:

```
%preproc(html): '^%html% ' ''
```

The leading string is removed and those lines will be "activated", not being comments anymore. As a explicit target config, this filter will be processed for HTML targets only.

Changing Txt2tags Marks with %!PreProc

Being a Regular Expressions guru, the user can customize the document source syntax, changing the txt2tags default marks to some he find more comfortable.

For example, a leading TAB is the Quotation mark. If the user doesn't like it, or his text editor has some strange relationship with TABs, he can define a new mark for Quoted text. Say a leading ">>> " was his choice. Then he will do this simple filter:

```
%!PreProc: '>>> ' '\t'
```

And on the document source, the quoted text will be something like:

```
>>> This is a quoted text.
>>> The user defined this strange mark.
>>> But they will be converted to TABs by PreProc.
```

Before the parsing begins, the strange ">>> " will be converted to TABs and txt2tags will recognize the Quote mark.

> **BEWARE!** Extreme PreProc rules could eventually change the entire marks syntax, even generating conflicts between marks. Be really really careful when doing this.

Part IX − Txt2tags HISTORY

On July 2001, it was launched the first public release of txt2tags, version 0.1. But its origins date more than a year before that...

This chapter gives an overview about the tool development since its very first draw until the current series.

1999 January: Pre−History

From the author:

> *"My really first attempts of a text conversion tool began back in 1999, as a very simple and limited Bourne Shell script that converts marked text to an HTML page. Yes, Yet−Another txt2html tool. Everyone, everywhere must have done one of this already... In short, it just recognized simple marks as `*bold*`, `/italic/`, `_under_`, and escape the classic `< & >` HTML special characters. Not impressive, but hey! I was young ;)"*

1999 June: Still Pre−History

The author wants to speak some more:

> *"Some months passed, and a big Sgml hype arrived at the company I was working (Conectiva). So the txt2html turned into a txt2sgml script. I was really trying to learn about SED* at that moment so txt2sgml was a 110 lines Bourne Shell script with lots of SED code."*
>
> * **SED:** UNIX Stream EDitor − an automatic text editing tool

This improved Sgml version has more supported structures as lists and verbatim text. On the following sample file, you can see the txt2tags marks origins:

```
* This was a bold line (BOLD line oriented? Well...)

  ––
− bullet list was very similar to txt2tags list
− but with these –– to begin and close a list
  ––

=–––––––––––––––––––––
Verbatim text was delimited by the =–– pattern.
The other ––––––– were just cosmetic.
=–––––––––––––––––––––
```

Still not impressive, but the big step is coming...

2000 August: Not Pre−History Anymore

The author once again:

> *"A year passed and at that time I was really in love with SED. The txt2sgml.sh shell script was rewritten and became a 350 lines pure SED script. Some exciting features were added as subsections, URL recognition and sublists. I have used and improved*

this script a lot, for almost a year."

A txt2sgml.sed sample file:

```
* Hey, here are the first 3 magic lines
* The document title / author / date
* But they required those asterisks at the beginning

MAIN TITLE

Titles were made by uppercase−only lines. Subtitles were identified by
leading spaces. Each space denoted a new sublevel. The beautifiers:
*bold*, **strong**, "italic" and `typewriter`.

− lists
  + sublists
    = and subsublists (by identifier, not indentation)


Two blank lines to close lists. Links as www.example.com and e−mails
were recognized automagically by regular expressions. And there was a
strange image mark:
%%image: path/to/image.jpg
```

2001 May: Pythonization and Multi−target Idea

Guess who will speak:

> *"I've started to write my Regular Expression book and used the txt2sgml.sed marked text format. This way I could convert it to sgml (then to HTML using sgml2html tool) and quickly check on the browser how the book was going. As 'quick' and 'sgml2html' don't match, I've modified the SED script to a txt2html.sed tool, generating HTML directly. [...] The publisher used Adobe PageMaker software to format books and it was a problem for a Linux guy like me. But I was happy to know that PageMaker had a tagged HTML−like language, so I've started to turn my script into a txt2pagemaker.sed tool. I've ended up with three similar SED scripts, converting my texts to sgml, HTML and PageMaker. And some other Shell scripts were made to extract the book TOC (Table Of Contents) and do post formatting. At the middle of the book writing, I had the idea to join it all in a single tool, and choose Python as the language. TXT2TAGS was born."*

2001 July: 0.x series: Debut of txt2tags (World Release)

Yes, him:

> *"The release of the printed book (July, 31) and the release of the very first txt2tags 0.1 version (July, 26) were very close, a matter of days. One depended on the other, they were developed together. Besides sgml, HTML and PageMaker, other targets were implemented for this release: MoinMoin, Magic Point and plain text. For more than a year, more releases of the 0.x series were made and the program began to grow: new target UNIX Man Page, %%date macro, table support, Web interface, smart image alignment and Table Of Contents generation. On version 0.2 a Shell script wrapper was added to handle file operation and command line options. I've made it because I was a shell wizard and a Python newbie. Only in version 0.9 txt2tags came back to a 100% Python code."*

2002 September: 1.x series: Growing

Won't you release the microphone?

> *"The txt2tags idea has proved to be good. I've decided to get serious on it. The next step was to spread the program out, to tell the world about it: documentation! The program site was launched, mailing lists (english and portuguese) were configured and the User Guide has born. The user base grew and many contributions came. Yes, I thought, it is working. The new features added on the 1.x series include: Graphical interface (GUI), Windows & Mac compatibility, LaTeX target, %!style, include command and the powerful Pre and Post processing filters."*

2004 July: 2.x series: Maturing

Ok, talk, I give up:

> *"Growing is hard and strange. Remember I said I was a Python newbie? Now I got better, but the old mistakes accumulated, and a major code rewrite was unavoidable. This broke backwards compatibility with some marks, and an upgrade script was created. It took a long time, but version 2.0 was released. It came with tons of news such as XHTML target, W3C validated code, i18n, and RC file. A team of translators has came and the program and its documentation were translated to many languages. The Lout target was added, and new macros: %%mtime, %%infile, %%outfile, and %%toc. Continues..."*

The End.