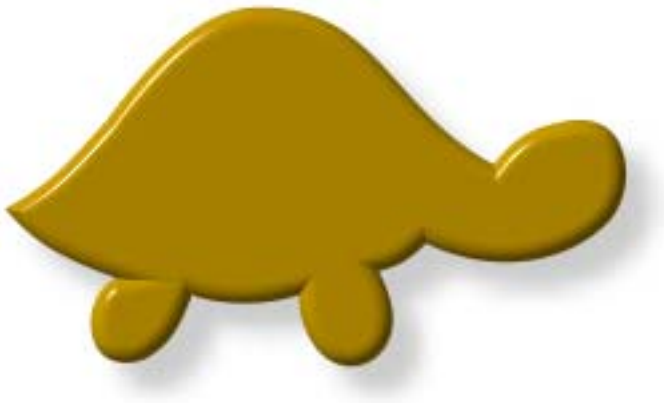
Tortoise



Tagger

ReadMe

Never be afraid to try something new.
Remember that a lone amateur built the Ark.
A large group of professionals built the Titanic.

General

Tortoise tagger is a Word template which can read data from a taglist created by the user and tag the text in the document according to it. Tagging is a series of Word's find/replace passes which a) finds and replaces strings of text and/or b) applies formatting to the text in the document.
Tagging is done on a copy of your plain text file which is then saved as a Word document. When translation has been completed, and the file has been cleaned, you need wither to copy/paste its contents into the original or save Word document as text and change its extension. All formatting is lost when you do it, and the translated file can be correctly interpreted by the respective application.

What are tags?

Tags are strings placed in the fabric of a document which control the final output created by a computer program: DTP or typesetting application, Web browser or anything else. A common example of tagged text is any web page, a line like this:

Click here for details.

is actually a chunk of HTML code, which your browser understands, and it looks like this:

```
<em><strong><font color="#0066FF" size="4" face="Verdana,
Arial, Helvetica, sans-serif">Click <a
href="http://www.accurussian.net">here</a> for
details.</font></strong></em>
```

where `<em>` stands for italics, `<strong>` makes the font bold, etc. Normally you should not know the meaning of every tag, but in order to make the proper word in the phrase a Web link you ***must know*** that it should be between `<a href="http://www.accurussian.net">` and `</a>`. Thus, you cannot avoid reading some reference on the issue, whatever format you are going to tag and translate.

Another less common example is LaTeX code, which brought about creation of the tagger. LaTeX is a highly sophisticated typesetting system with virtually unlimited capabilities and it can be extensively expanded and customized, probably that is why there are no LaTeX filters available. A brief example of LaTeX code is below.

```
\article
\head Introduction\endhead
A recent article in{\it
Time${}^{\kern4pt\reg}$\/}\vfootnote{\reg}{\ninerm{\it
Time\/} is a registered trademark of AOL Time Warner,~Inc.}
magazine's {\it On-Line\/} monthly ``submagazine'' explored
the world of do-it-yourself font creation and manipulation.
The orientation of the article was to help a relative novice
chose the right tools and techniques for whatever kind of
font work was desired. The article was heavy on facts
concerning a four-step process that might be familiar to
readers of \TUB: \list[\unitemized\numbered]
```

It is an extract from LaTeX source code found "TUG1.tex" file, its final output is in "TUG1.pdf" file.

There is another large file included in this package: 'beginlatex.pdf', which is a brief explanation of the format. It is included here for you to understand the logic of LaTeX taglist and, if necessary, to be able to modify it and/or create your own taglists.

Summing up the tags:

- tags are strings within a tagged document;
- tags are not visible in the output, but they fundamentally affect it;
- tags generally should not be translated, but some tags have to be repositioned to match the meaning of the original.

What is tagging?

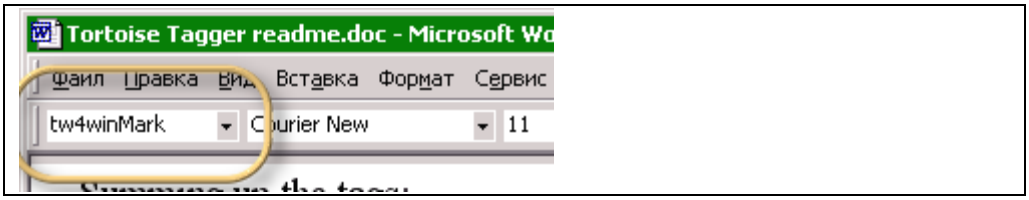
The process of tagging (as I understand it) means marking the tags known to the tagging utility with appropriate format, usually style.

Why style?

Wordfast, one of the leading CAT tools relies on MS Word styles while you are doing your translation. Normally you see only one 'special' style in a Word document: ***tw4winMark***, this is the style of the delimiters separating source and target segments. Here's an example:

{0>The answer was “yes”.<}0{>Ответ был утвердительным.<0}

The purple delimiters have this style. Wordfast can 'see' the delimiters just because they have this particular style. If you open any uncleaned file and move the cursor onto the "{0>", "<}0{>" or "<0}", you will see the name of the style in the 'formatting' toolbar's style name window, as below.



However, *tw4winMark* style is not used for tagging, the two styles which serve this purpose are *tw4winExternal* and *tw4winInternal* styles. Their names are quite self-explanatory: everything in *tw4winExternal* is bypassed during translation, everything *tw4winInternal* is included in the segments but is regarded as something unalterable. Wordfast has a built-in feature to make sure that the tags in the source and target segments are identical.

Here's a chunk of tagged Adobe InDesign code in *tw4winExternal*:

```
<Version:3.000000><FeatureSet:InDesign-
Japanese><ColorTable:=<Black:COLOR:CMYK:Process:0.000000
,0.000000,0.000000,1.000000>>
```

It is evident that nothing here should be translated.

Here's an example of a LaTeX source code with some parts in *tw4winInternal*:

```
\subsection{Ligatures}
In typography, a \emph{ligature} is a glyph which has been formed by
joining glyphs that represent two or more characters; this joining can involve
quite a lot of deformation of the original shapes.
```

It is a subsection header and a sentence from a paragraph. Here, the 'styled' commands must be together with the text you are translating. The header's target segment should contain all the tags existing in the source segment in the same order or following the same logic. The sentence should have the two tags inside it (opening – `\emph{` and closing – `}`) and they should be repositioned to enclose the word which stand for "ligature" in the target language. It's as simple as that.

The good thing about styles is that Wordfast and other CAT tools find them by their *name* only, which means that you can set your own parameters for the styles in your normal.dot global template, and the tagger will apply *your* styles' parameters to the tags. This helps to reduce eye strain and is not harmful to Wordfast performance and the final plain text output.

Another style which the tagger uses is 'Translatable'. Previous versions of the tagger used to mark the entire document as Translatable, then the tags were marked with the two special styles discussed earlier. However, it turned out that some translators need to perform formatting or replacement on already tagged *Word documents*, therefore, automatic formatting with Translatable was dropped, and a command *~~~DocTrbl* was introduced, which does just the same. Having Translatable style in a tagged document is useful, this makes it

easier to perform searches in the tagged document, especially when you are building your own taglist, because this enables you to look for, say, "=:>" string in any of the document's styles (see MS Word help for details).

If your normal.dot global template lacks these styles, Tortoise Tagger inserts them in *to the document* automatically. If you already have them and even if you customized them, the tagger uses *your* styles.

One last point. If, while translating with Wordfast, you need to apply 'Normal' style to any part of your document, select it and hit Ctrl+Space.

Word's Find / Replace

Basics

This feature of MS Word helps to find any string in the opened document and replace it with whatever you want it to. A simple example is find all "Manchester Polytechnic" in your CV and replace it with "Harvard University". In order to do this, you must have "Manchester Polytechnic" in 'Find What' field and "Harvard University" in 'Replace With' field. Pretty simple, isn't it?

However, there are more options in the F/R (find/replace) feature. If you click 'More' button, the dialog box will expand and you will see some tick boxes and buttons. In its operation, the tagger uses the standard Word F/R feature, supplying settings, find and replace stings to it, and instructing Word to execute a F/R pass with the assigned parameters. The feature is well described in Word's help system, on numerous Web-sites and far more numerous books. Only some aspects of the feature, relevant to the tagger's operation, are discussed here.

Wildcards

If you tick this box you will be able to use *masks* for search. For example, you would want to find all strings like 'That day Mary bought a pencil in the shop', where Mary bought a huge number of various things, and replace it with 'That day Mary went shopping again'.

To do this, you must type 'That day Mary bought * in the shop' in 'Find What" field and 'That day Mary went shopping again' in 'Replace With' field, where the asterisk '*' would stand for any number of any characters. Word will find all the phrases matching your criterion and will replace them with what you typed in 'Replace With' field.

IMPORTANT-1

Word's F/R feature operates on *lazy* principle, which means that Word stops looking for new matches as soon as the shortest one is found. Therefore, in a text like this:

```
It was a hot Alaskan December morning. That day Mary bought a
pencil in the shop. She used it to pick her nose and drew a
lot of pictures on the walls. Another day came. That day Mary
bought a hammer in the shop. She couldn't pick her nose and
smashed the furniture in despair.
```

you will have matches found like these:

```
It was a hot Alaskan December morning. That day Mary bought a
pencil in the shop. She used it to pick her nose and drew a
lot of pictures on the walls. Another day came. That day Mary
bought a hammer in the shop. She couldn't pick her nose and
smashed the furniture in despair.
```

rather than this:

```
It was a hot Alaskan December morning. That day Mary bought a
pencil in the shop. She used it to pick her nose and drew a
lot of pictures on the walls. Another day came. That day Mary
bought a hammer in the shop. She couldn't pick her nose and
smashed the furniture in despair.
```

although the last match *formally* fits your search criterion: It starts with '*That day Mary bought*' has many other characters in the middle and ends with '*in the shop*'

This makes it easy for user to make appropriate Find What strings, like the one from LaTeX taglist:

```
\\begin\{verbatim\}*\\end\{verbatim\}
```

because, despite the fact that there are plenty of such command pairs in most LaTeX documents, Word will find the closest ones, the opening and closing tags, exactly what you need.

Question mark '?' substitutes any *single* character in wildcard mode.

IMPORTANT-2

It should be mentioned that if you have a short closing string consisting of 1 or two characters, especially the ones used to set advanced FR options, asterisks should be avoided at all costs. When you need to tag a string like

```
String <`Translatable text here.'>
```

you should NEVER make a mask like this

```
String \<`*'\>
```

because for reasons I do not know, Word will go comatose when you run the search. Instead, use this mask

```
String <`[!\>]@\>
```

This produces reliable results.

Backslash and a few other odd characters

"Why does the example above contain so many backslashes?" you would ask. This is because with 'match wildcards' mode activated, you cannot type certain characters as they are, but have to type a backslash before them, to tell Word that they are just characters, and not delimiters in your F/R input. If you need to find a backslash in 'match wildcards' mode, you should type another backslash before it.

Other characters which must have a backslash before them in 'match wildcards' mode are as follows, their 'wildcard' mode meaning is specified too:

{} – used to specify the number of character repetitions;
[] – used to specify character ranges;
* – stands for any number of any characters;
? – stands for any single character;
! – stands for 'except' or 'not';
@ – stands for 'any number' of the preceding character or range;
(and) – used to split the Find What field contents into groups;
<and > – used to specify the beginning and the end of a word

Hard return and the like

Very often you need to specify non-printable characters in F/R fields. In simple mode the F/R dialog itself offers you a ready-made collection of those, which you select from a drop-down list, but they do not work in 'match wildcards' mode. Therefore, those few must be specified using their numeric code:

tab mark – ^0009 or ^9;
line break – ^11;
page break – ^12;
hard return – ^13;
column break – ^14;
long dash – ^30;
space – ^32.

In many cases for 'space' you may either use the code or press space, but pressing the spacebar has a great disadvantage: *you don't see* it in the taglist.

Here are a few examples from the taglist:

```
\\begin\{verbatim\}*\\end\{verbatim\}
```

which means: "find in wildcard mode everything that begins with `\begin{verbatim}`, contains any number of characters and ends with `\end{verbatim}`.

```
\\verbatim[^13]*\\endverbatim[^13]
```

which means: "find in wildcard mode everything that begins with `\verbatim + hard return`, contains any number of characters and ends with `\endverbatim + hard return`

```
%[!^13]@^13
```

which means: "find in wildcard mode everything that begins with `%` (per cent sign), contains `any number of characters other than a hard return` and ends with `a hard return`

```
([!^13])^13([!^13]) \1 \2
```

which means: ***find*** in wildcard mode everything that begins with any single character other than a hard return, a hard return and ends with any single character other than a hard return;

replace it with what you have in the first brackets, a space and what you have in the second brackets.

(this pass replaces single hard returns with spaces)

Formatting

If you invoke the dialog and run a F/R pass with 'Replace With' field empty you will delete from the document whatever is specified in the 'Find What' field. However, if you place the cursor in the 'Find What' field, click 'more' button, 'format' button and select any format, instead of deleting the text Word will format it accordingly. The tagger uses this technique to apply styles and other attributes to the text in the document.

Sometimes you need to delete *some* of the characters leaving the rest in the document. This happens in the LaTeX tagging sequence when some hard returns are deleted by first bolding the those which must be kept (in comments, verbatim and tabbing passages etc.) and then removing the hard returns which are not bold. This may be achieved by setting *not bold* in 'more – format – font' dialog of the F/R control box. Distinguishing between the same 'needed' and 'disposable' characters can be done using styles or font colour too, but to me using bold attribute was simpler.

If you start making your own taglist, remember that the tagger simply supplies parameters and strings to Word's F/R dialog, so you may try your variants of

strings and parameters 'by hand', using the F/R dialog first and see if it produces intended results.

Installing Tortoise Tagger

Unzip it from the package and copy in a folder of your choice. Start Word, select 'tools – add-ins', click on 'add' button, navigate to the folder and select TortoiseTagger.dot file, the tagger will appear in the list of add-ins. Check the box next to the tagger and close the dialog.

If you want the tagger to be active every time you start Word, copy it to Word or MS Office startup folder (search for 'startup' on your hard drive).

You can simply open the Tagger as an ordinary document, click on 'enable macros' when prompted and use it. It will be disabled as soon as you close it. Remember not to save any changes to it then.

If you don't see the tagger's toolbar, go to 'view – toolbars' and select 'Tortoise Tagger'. The toolbar looks like this:

you can dock it, if you wish.

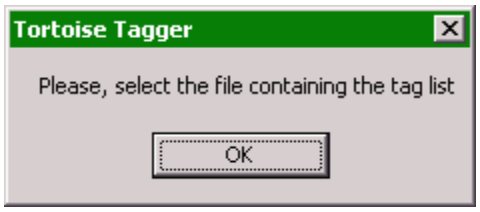
The button with a running tortoise performs the tagging, the button with somebody's left eye reveals spaces, hard return, hidden text etc., the button with footprints hides everything but printable text. Lastly, the button with the question mark displays an info box with lots of valuable info (my credit card number and PIN, among other things).

Tagging

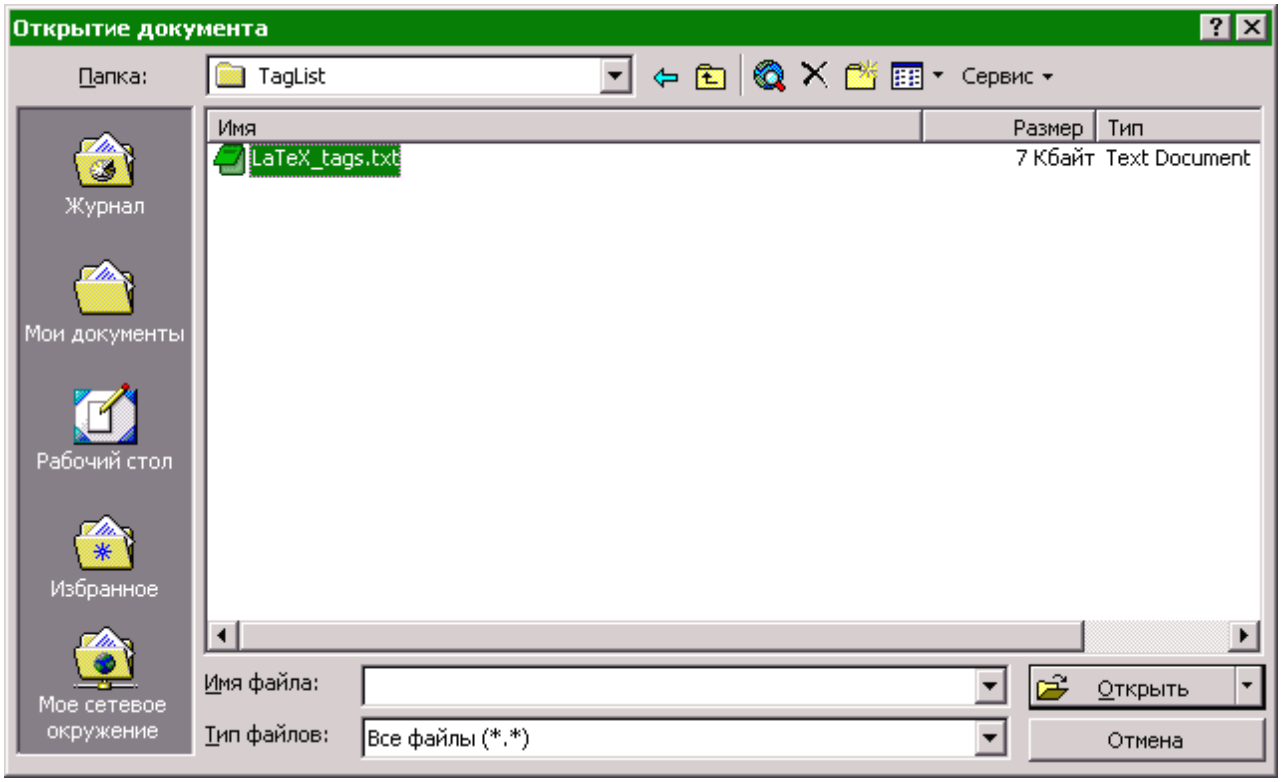
In order to tag you need a taglist file and one or more workfiles. A taglist file for LaTeX format is supplied with the package, so are a few LaTeX source code files. It is recommended to keep taglists in text format, to make it easier to edit them in Notepad or Word, the only restriction about workfiles is that they must have an extension, because the tagger runs in batch mode, processing all the files of the same type in the current folder. If you point the tagger to a workfile without an extension it will refuse to work. If your files are without an extension, you must temporarily rename them.

When you run the tagger, you point to the taglist and the workfiles in a standard Word dialog. If you click 'cancel' in any of the dialogs, the tagger aborts.

Click the 'TAG' button. A message box will pop up, reminding you what you should do:



and when you click 'OK', a dialog will open, where you must navigate to the taglist and select it by double clicking or hitting 'Enter'.



Once the taglist has been selected, you will be prompted to point to one of the workfiles in similar manner. You may store your taglists and workfiles wherever you wish, together or separately.

Tagging is done in batch mode on copies of your original files. The tagger opens the workfiles, performs tagging and saves them as documents, including the original extension in the filename.

One more point is that Tortoise Tagger is a foolish program and every time you point it to a plain text workfile it creates and saves a Word document for your workfile, **overwriting any existing Word document.** A warning dialog reminds you of this, because you might ruin already translated files otherwise.

Taglist syntax

Comments

An option is provided to include comments in the taglist. Since the taglist is a small computer program, it is a good idea to make notes regarding what this or that line stands for, because with time you may well forget the details.

A comment is a line beginning with *3 per cent signs in a row*: %%%. You *cannot* start comments in the same line, *after* the commands. generally speaking, you may simply type comments into the taglist without any per cent signs, because chances are next to nothing that there will be the same line in the document you are about to tag, but you never know, and, as it usually happens, you may have unexpected results when tagging a new file some six later, when you completely forgot that you added a comment without '%%%'. Another thing about the 3 per cent signs is that when the tagger encounters them, it skips the rest of the processing mechanics, which is a split second faster than using the comment as a F/R string, but it may be noticeable when you tag a few hundred long files. So, these are simply manifestations of my efforts to combat sclerosis:

```
%%% bolding starts here ---
```

Commands

All Tortoise Tagger commands begin with 3 tildes and end with a hard return, the best way to avoid trouble is to store them all in the taglist and copy/paste them to any point of the list. If the tagger encounters a mistyped command beginning with '~~~' it will warn you. If a tilde is missing, the tagger assumes it is a string, and the result is - your mistyped command is not executed and used as a string in Find What field.

The commands are fairly self-explanatory. Here's the complete list of Tortoise Tagger commands:

~~FindBold	– search for bold text;
~~WriteBold	– make the replacement text bold ;
~~FindNotBold	– search for text which is not bold;
~~WriteNotBold	– make the replacement text not bold;
~~FindAsIs	– search for any text, irrespective of its format;
~~WriteAsIs	– replace the text as it is, irrespective of its format;
~~FindInternal	– search for text in *tw4winInternal* style;
~~WriteInternal	– apply *tw4winInternal* style to replacement
~~FindExternal	– search for text in *tw4winExternal* style;
~~WriteExternal	– apply *tw4winExternal* style to replacement;
~~FindTrbl	– find text with 'translatable attribute. The style may either be present in your 'normal.dot' template or defined by Tortoise Tagger. It is worth while remembering that at the beginning the tagger makes *entire* document translatable;

~~WriteTrbl	– apply translatable style to replacement. Sometimes it is easier or faster to make entire document *tw4winInternal* or *tw4winExternal* or hidden (for DejaVu users) and then expose the lesser part for translation, like in Frame Maker's MIF document, where most of the code is not for translation (see below ~~~Doc* commands;
~~FindHidden	– search for hidden text;
~~WriteHidden	– make the replacement text hidden;
~~WC-ON	– activate 'match wildcards' mode;
~~WC-OFF	– deactivate 'match wildcards' mode;
~~FindHilite	– search for text with *any* highlighting;
~~WriteHilite	– make the replacement text highlighted[1];
~~FindDStrike	– search for text with double strike through attribute;
~~WriteDStrike	– make the replacement text double strike through[2];
~~Case-ON	– activate 'match case' mode;
~~Case-OFF	– deactivate 'match case' mode;
~~HWord-ON	– activate 'match whole word' mode;
~~HWord-OFF	– deactivate 'match whole word' mode;
~~DocInt	– apply *tw4winInternal* to entire document. Useful when the translatable text makes a small portion in the document and falls into a simple pattern which can be implemented in one or several passes;
~~DocExt	– same as above but with *tw4winExternal*;
~~DocBold	– bolds all text in the document;
~~DocUnbold	– remove bold attribute from all text in the document;
~~DocHide	– makes all text in the document hidden;
~~DocUnhide	– remove hidden attribute from all text in the document;
~~DocTrbl	– makes all text in the document translatable;
~~Demo	– a silly command, which turns on updating of Word's screen while the tagger is buzzing, so you can see with your own eyes what is happening behind the glass of your monitor;
~~Stop	– stops processing of the file, displays an info message containing data useful for finding logical errors in your taglist, and writes a copy of this message into a file.

This last '~~~Stop' command is useful when the tagger runs fine but you have unexpected results. Normally the culprit hides somewhere in the middle of the taglist, for example you wrongly assume that the string you're formatting has one style, while it has been fully or partially formatted with another. Of course, you can delete or disable strings and settings in the taglist by commenting them out, but in this case the tagger sequence will terminate *normally*, without displaying the debugging message, which has proven quite useful when it comes to tracing a *logical* error in the taglist.

The commands applying 'hidden' attribute are not used when tagging files for Wordfast, this option has been added taking into account some wise advice from the friendly camp of DejaVu user list (see below).

[1] Highlighting with 25% grey is one of the choices that Wordfast gives you for marking text as untranslatable (you should set it in Wordfast settings). Initially the idea was to use this particular shading, but, my inexperience in VBA (I could not write it properly) and common sense (I still have extremely brief flashes of this) told me to leave it just 'highlighting'. In order to set it 25% grey you must select this highlighting in one of Word's toolbars before you run the tagger. The advantage of this is that you can now highlight any text in any number of documents with any of the available colours. Well, it does not sound big deal to anyone *who does not have to do it*.

[2] One of Wordfast choices of untranslatable attribute. Marching Red Ants are not available.

Singles / Doubles

If you need to delete or format strings in the document, you specify a string in the taglist, which ends with a hard return. This is called 'a single'. If you actually have something to offer to the tagger for replacement, like find 'Banana' and replace it with 'Bamboo', you must type 'Banana', press a tab, and type 'Bamboo'. The tagger treats tab-delimited strings as two strings, the first part goes to 'Find What' field, the second - to 'Replace With' field.

LaTeX taglist explained

Let us now follow the taglist included into the package. It contains comments which I made for myself and for those who would read or edit the taglist. The commands are not explained here because they have already been covered.

If you review the examples of the LaTeX files in the package you will see that their creators use hard returns to start a new line and use many spaces to indent the text and commands. This is done to improve readability, but it will interfere with translation: hard returns will cause segmentation problems, multiple spaces will cause troubles with Wordfast's Quality Check (if it's activated) and/or result in eye strain.

The first section of the taglist makes bold the parts of the document where paragraph marks must not be deleted, because they either form the logical structure of a list or table, or are comments (if we delete a paragraph mark ending a comment line, it will disable everything that follows, till the next hard return). Please, pay attention to the fact that every new pass looks for not bold text. This is very important, as well as the sequence of the lines in the bolding section.

Bolding 'good' paragraphs

The 4 lines below bold the text where each and every character should be kept.

```
\\begin\{verbatim\}*\\end\{verbatim\}
\\verbatim[^13]*\\endverbatim[^13]
\\begin\{tabbing\}*\\end\{tabbing\}
\\tabbing*\\endtabbing
```

This line below bolds a backslash and a per cent sign.

```
\%
```

This is a combination of characters used in LaTeX to denote a per cent sign. If this is not done, when we bold comments, the tagger would find this per cent sign, look for the nearest paragraph mark and make it a comment, while the '\%' combination is nearly always a part of normal text.

The two lines below bold a) lines beginning with '%' and ending with a paragraph mark, i.e. lines of comments at the beginning and in the middle of a line; b) a '%' ending a line.

```
%[!^13]@^13
%^13
```

This is necessary to prevent comments from being incorporated into the rest of the text, effectively disabling everything that follows till the nearest paragraph mark.

Removing 'bad' paragraphs and multiple spaces

The next section of the taglist consists of doubles, because in this section we look for single paragraph marks:

```
([!^13])^13([!^13]) \1 \2
```

and replace them with spaces.

Find multiple spaces

```
[^32]{2;}
```

and replace then with one space.

IMPORTANT NOTES:

1) In the above line there is a tab an a space after the closing curly bracket
2) On some local versions you must have a comma "," instead of the semicolon in the string: [^32]{2,}

The end of previous line, a space at the beginning of the new line and a non-whitespace character are found

```
([^13])^32([!32])    \1\2
```

and then replaced with the same, but without the space.

Style 'sure LaTeX' strings

Then there is a pass applying internal style to the strings which I spotted inside the text and in 'beginlatex.pdf' document. Wildcards are off.

```
\end{quote}
$\{$
$\}$
$\backslash$
$*$
+ many more ...
```

There was a small problem with LIG string, because its inclusion into the taglist at an earlier stage of the tagger development resulted in the tagging of all occurrences of the string, like slight etc. Therefore, this string is tagged with 'Match case' parameter active.

Style LaTeX commands with Wildcards

Now, very often it is hard or inexpedient to have all the strings as they are in the code, because many of them fall into a pattern, which you can record as a single string and run a pass using wildcards.

The string below covers most LaTeX commands, even those, which I don't know about, because it uses wildcards.

```
\\[!^0013^0009^0032\{=>,\)]@[^0013^0009^0032\{]
```

Let me decipher it for you.

Find

\\	- anything that begins with a backslash
[	- range starts
!	- does not contain
^0013^0009^0032\{=>,\)	- these (a paragraph mark, a tab, a space, an opening curly, an equal sign, a comma, a closing bracket)
]	- range ends
@	- any number of them, and ends with
[	- range starts
^0013^0009^0032\{	- any of these (a paragraph mark, a tab, a space)
]	- range ends

The string described above used to look like this:

```
\\[!^0013^0009^0032\{]@[^0013^0009^0032\{]
```

but then I saw some erroneously tagged strings in the document and added these:

```
=>,\)
```


The tagged strings which you want plain are very easy to spot: they are marked red in the text and you will have no problem detecting wrongly tagged strings and making appropriate amendments to the taglist. If you want *to exclude* anything from the tagged sting, add it to the first range, if you need to add a character you want the string to end with – it should go into the second range.

The following line

```
\\[a-z]@=
```

tags anything starting with a backslash, any number of lowercase English letters from 'a' to 'z' and ending with an equal sign. This string contains a possible flaw - your developer may have other, 'local' characters in the commands, which would exclude the commands containing the local characters from the 'Find What' range. At the same time local versions of LaTeX are sometimes easier to tag. A good example is a typical command from a job I did some time ago, which looked like this:

```
\somecommand{arg1}{arg2}{RussianText}
```

Therefore, I could tag everything not in Russian [!А-я] between curly brackets as tags and forget about them altogether.

I skip some strings here and proceed to the apostrophe. In LaTeX quotes are made like this ``quotes'' (double) or this `quotes' (single). Please, note that the closing quote in the double quotes consists of 2 apostrophes. The task was to distinguish between the apostrophe in, say 'don't' and the apostrophe comprising those LaTeX quotes. My solution (listing all possible variants) is far from elegant, but it works, if you have a better solution, please, share it with other folks and drop me a line.

Literal pass with external style.

I just dumped a chunk of LaTeX code into the taglist to see what happens (see the taglist). The external style, let me repeat, is completely bypassed by Wordfast, hence the name, but its use, however tempting, has some pitfalls if you translate a LaTeX document about LaTeX. :) In this case you may have examples of commands within the fabric of the text, and they will, of course, break your sentences, making it very hard for you to translate them, so, please, keep it in mind.

Wildcards pass with external style

Just some of the strings. The principle is the same as with internal styling.

```
\{?.??\\textwidth\}
\\penalty-[0-9]@ \}
```

`?.??` - any character, a period, any 2 characters
`[0-9]@` - any number of digits from 0 to 9

Straighten LaTeX lists

And lastly, the final touch. The \item command in LaTeX is used to create lists. The previous passes deleting single paragraphs ruin the list structure made by the authors, which, I think, should be restored, for better readability, which will also facilitate translation a bit. This line is tabbed!

```
\item    ^p\item
```

`\item` - find this
`^p\item` - replace with a paragraph mark and the same string.

It's like manually hitting 'Enter" at every `\item` string. Of course, there is a more elegant syntax with wildcards activated:

```
(\item) ^13\1
```

but when I was making the taglist I was thinking about readability and ease of understanding first.

That is it with LaTeX taglist. Please, keep in mind that my LaTeX taglist is not comprehensive, that is the main reason why the F/R parameters and strings have been moved out of the program code, and user is now able to customize every F/R parameter.

More examples

Please, check http://www.accurussian.net/tagger.htm for more file formats that the tagger can process.

The examples below are for the sake of illustrating how seemingly very complex tasks may be solved using simple procedures.

As of today, I have reports of jobs completed using the tagger, for all formats described in the manual.

Adobe InDesign

As of today I have successfully completed a practical assignment translating a user manual for a DVD system, a heavily formatted DTP job with complex structure. The files I received from the client were output of Trados Story Collector, these are plain text files with .ISC extension. If you open one such file you will see something like that:

```
<TRADOSStoryCollector SCVersion="6.5" DTPPackage="InDesign"
DTPPackageVersion="2" Encoding="UNICODE"><STORY NAME="1"
LOCATION="MB-master2"><UNICODE-WIN>
<Version:3.000000><FeatureSet:InDesign-
Japanese><ColorTable:=<Black:COLOR:CMYK:Process:0.000000,0.00
0000,0.000000,1.000000>>
<DefineParaStyle:08R=<Nextstyle:08R><cSize:8.000000><cAutoPai
rKern:None><cKerning:0.000000><cLeading:10.000000><cLanguage:
Neutral><pHyphenation:0><cFont:Arial><cHang:Baseline>>
<DefineParaStyle:07R=<BasedOn:08R><Nextstyle:07R><cSize:7.000
000><cLeading:9.000000><cLanguage:Neutral><cFont:Arial><pText
Alignment:Left>>
<DefineParaStyle:06.5R=<BasedOn:07R><Nextstyle:06.5R><cSize:6
.500000><cLanguage:Neutral><cFont:Arial>>
<ParaStyle:06.5R>RQT7937
</STORY>
```

A simple analysis shows that everything you need to translate is ***not*** between '<' and '>'. However, some of the tags (strings between '<' and '>') do occur within a sentence. The taglist is available from Tortoise tagger download page.

Let me comment it a bit since with every particular job the taglist will probably need a bit of tweaking (that's what Tortoise Tagger was created for in the first place):

```
~~~WC-ON
~~~FindNotBold
~~~WriteBold
\<CharStyle:*\>
```

This section bolds tags which are used to format text, very often they occur within a sentence. Bolding is applied in order to subsequently allow the tagger distinguish between the tags which should be tagged external and those which should not (i.e. bold).

```
~~~FindNotBold
~~~WriteExternal
\<[!\>]@[\>]@
```

This pass (remember that 'match wildcards is still on!) applies external style to everything between '<' and '>' but bold.

```
~~~FindBold
~~~WriteInternal
*
```

This pass ('match wildcards is still on!) finds everything (anything, if you like) which we bolded in the beginning, remember?

```
~~~WC-OFF
~~~FindAsIs
<ParaStyle:07R-dot>
<ParaStyle:07R>
<ParaStyle:08B>
```

Here is the place where you will probably do all the tweaking. These are *real* styles from *my* job. My client informed me that these occur inside a sentence, therefore, styling them external would have resulted in segmentation problems. Luckily, it was possible to limit the number of these tags to just three. Please, note that 'match wildcards' is disabled, in order to make it easy for me to copy/paste them from the document.

Keep in mind:
If you do not have data on such tags, i.e. those occurring in the sentence, it would be a good idea to comment all the lines in this section and tag JUST ONE FILE. Then you will go on translating the document, when you encounter a tag which breaks a sentence, you should:
a) move it to the discussed section of the taglist and save the taglist;
b) perform a find/replace on the document you are translating, putting the tag in the 'FindWhat' field and leaving 'ReplaceWith' blank and specifying tw4WinInternal style in 'more-format-style' pane of the F/R dialog.
Of course, you can simply retag the file with amended taglist and then translate it from the TM, but that is a bit boring.

And, lastly,

```
~~~FindAsIs
~~~WriteExternal
>>
```

There were some '>' characters in the file which remained not tagged since the tagger was looking for the *first* closing 'greater-than'. Leaving them as they were was not lethal but my eye kept on stumbling upon them and I added this

line, specifying two such characters because I had noticed that they never occurred together in the tags within sentences.

InDesign Workflow

Tag - translate - clean - save as text - change file extension to ISC.

Quark Express

Once again this is output of Trados Story Collector,

The beginning of a Quark Express file looks like this:

```
<$XPExtTags 1.0 win><STORY NAME="6" LOCATION="P1"><v3.01><e1>
@Normal=<Ps100t0h100z12k0b0cKf"ArialMT">
@Normal=[S"","Normal","Normal"]<*L*h"Standard"*kn0*kt0*ra0*rb
0*d0*p(0,0,0,0,0,0,g,"U.S. English")>
@$:<*p(0,0,0,0,0,0,g,"U.S. English")>This is a story about a
translator who is <snip>
```

Similarly to InDesign story file most of the stuff between the '<' and '>' or between '@' and ')>' should be left outside translation. However, a few tags are within the fabric of the text. Hence the taglist (check downloads page).

I created a simple taglist myself, but the list available on downloads page has been created by Nicolas Racine, a freelance translator, who added a lot of tags, straightened the taglist structure and used it for tagging.

'Match wildcards' is active throughout the list, except for </STORY> tag. First, everything between and including '<STORY' and ')>' is marked external (\<STORY*\)\>), most of the tags are between '@$' and ')>' or '<' and '>' (\@$*\)\>) and (\<[!\>]@\>), respectively, and then tags, seemingly responsible for character styles in the sentences are marked internal, to allow Wordfast include them in the segment, opening tags are reduced to anything between '<c"' and '">' (\<c"*"\>) and closing tags seemingly are all <c$> (\<c$\>).

Quark Express Workflow

Tag - translate - clean - save as text - change file extension to QSC.

Frame Maker MIF file

Frame Maker is capable of saving entire file as plain text with tags. The files are quite large and most of the data is tags. My advice is to ask your client to break the publication into small parts, because Word has troubles handling files of several megabytes in size (5-10 pages in Frame Maker).

IMPORTANT
1 --- Unlike other formats, MIF files require TWO passes: tagging and untagging.
2 --- Tagging and untagging passes include font mapping. At present only English to Russian and Polish taglists are available. However, font mapping part is very easy, and tagging part is nearly the same for all languages.

Frame Maker MIF Workflow

Tag - translate - clean - untag Word docs - save as text - rename to *.MIF.

More info on font mapping will be available when time permits.

Game resource file

Another example is taken from a forum post, a fellow translator was asking for help with the following (most probably this was a resource file for a shooter game):

```
 {TEXT("QUIT GAME")},
 {TEXT("BACK")},

 {TEXT("OBJECTIVES")},
 {TEXT("Guide our hero around each level,\npainting all the
blocks to the\nrequired color.  Avoid contact with\nthe
enemies at all costs.")},
 {TEXT("Use the lifts if things are getting\ntough.  Simply
jump onto them and\nyou will be taken to the top of
the\nlevel.")},
```

Here I assumed the \n is a newline character and first padded it from the rest of the text with spaces and then applied internal style to it. The rest is tagged external, because they do not interfere with the sentence structure. The taglist is in 'game_msg_tags.txt' file. Source and the tagged output are in the same folder.

After translation and cleanup the file must be processed again in order to delete spaces around the '\n', which can be done "by hand' or, better, with Tortoise tagger again, because it can process files in batch mode.

Web Database file

The file is as follows:

```
Props 161 200   Front Seat: with Separate Headrest
Props 161 200
Props 161 200

Props 161 298   Back Door: Removable
Props 161 298
Props 161 298

ConfigGroups 0 42Annual subscription (show cross-reference)
ConfigGroups 0 42Annual subscription (show cross-reference)
ConfigGroups 0 42

ConfigGroups 0 46Payment per counter - monthly - suppliers
only (no labor information), show cross-reference
```


The taglist is like this:

```
~~~WC-ON
~~~WriteExternal
Props [0-9]@ [0-9]@
ConfigGroups [0-9]@ [0-9]{1;}
^13[0-9]{1;}
```

No more comments are required I guess.

Translating tagged documents

The approach should be quite the same as to any conventional tagged document: it is recommended to activate Wordfast's Quality Check and instruct it to ensure identical tags in the source and target segments.

Once again, you should know what the text formatting tags look like in order to be able to reposition them according to the sentence structure of your translation.

Saving your output

Since plain text files are incapable of preserving any formatting you can either save your cleaned Word document as plain text or copy its contents, paste into Notepad and save with an appropriate extension.

One point to observe if you would use *hidden text* for tagging: since hidden text is not copied into Windows clipboard, prior to copying it you should remove this attribute from all the text in the document. This can be done "by hand', with standard Word's 'font' dialog (Ctrl+D) or with the tagger, the latter option is reasonable if you have many files to process and/or need to perform some additional post-translation processing.

Making your own taglist

Once again, you should clearly understand which tags are always outside sentences and which are always or often inside them, the former may be tagged `tw4winExternal`, the latter must be tagged `tw4winInternal`.

A good idea is to open one of the longest workfiles in Word, delete all text and put all commands etc. in one column. Then you can either sort them in MS Excel or save the document as a text file and sort it using Wordfast glossary reorganise feature. This way it will be easier for you to see the pattern the commands fall into, create 'wildcarded' strings which would cover much of the commands, most probably even those which are in the other workfiles which you haven't reviewed yet. The top part of LaTeX and similar files usually contain things for the compiler which are not to be translated, therefore, copy/pasting them into the 'external' section of the taglist may be a practical approach, and then splitting those into 'wildcarded' and 'literal'.

Avoid setting long find strings, because, at least on my system, Word stumbles on things like I once offered it in a Frame Maker *.mif file: quite logically I wanted the tagger to make *tw4winExternal* everything from the beginning of the document to the first <PgfTag `Body'> tag, because in my example file all the text was after that tag. I should have noticed that there were some 14 thousand lines in this mask. MS Word kept me waiting for a few minutes, then I hit 'Escape' to interrupt the F/R process and saw that the section of the document had actually been tagged external, but the process had halted somehow. This might work on faster machines, though.

In order to debug your taglist you can use the '~~~stop' command, which will halt tagging midway, display current settings to you and leave the document

open, allowing you to review the results. A bit of trial and error and reading some Word online tutorials will get you going.

There is a chance that in wildcard mode you may specify a string which is illegal for it, for example '^p'. This string will generate an error and a message box will be displayed with all the current settings for you to make your conclusions. Since it is not possible to learn by heart or copy and paste the contents of the message box, the tagger also writes all the settings into a 'dammit.txt' dump file which you will find in the workfiles folder, review and adjust the taglist respectively.

Since Tortoise Tagger is still a newborn baby, there is little feedback from the users for me to analyse and include in this manual. If you experience problems or found a solution you'd like to share, please, drop me a message at ttagger@accurussian.net. I would also be grateful if you submit your taglist(s) with comments and your name inside for me to post and for other fellow translators to use. The code of the program has proven to be bullet-proof, therefore, all the research and tweaking is about file formats and taglists, where anyone can achieve positive results.

If you need to make something internal & hide it you have to make 2 passes and not 2, for example (quite a silly one by the way), if in the attached game resource file you would want to first apply *tw4winInternal* to the newline character '\n' and then hide it from view, you must make the list as follows:

```
~~~WriteInternal
\n
~~~WriteHidden
\n
```

and NOT

```
~~~WriteInternal
~~~WriteHidden
\n
```

because the second 'Write' command will effectively disable the previous one.

Do not leave empty lines in the taglist. It has been noticed to affect the tagging results, I did not establish the pattern, but empty lines have negative effect upon Tortoise Tagger. If you want to separate sections of the tag clusters, please, use comments.

Trados compatibility

I briefly tried a LaTeX file in Word with Trados (not in Tag Editor). Trados runs smoothly, but lacks Wordfast's Quality Check, therefore you should pay extra attention to keeping identical tags in source and target segments.

DejaVu compatibility

Following useful feedback by the members of DejaVu Yahoo group, a change was made to enable Tortoise Tagger to format text strings as hidden text, because, according to the opinion of experienced DejaVu users, this approach is the most practical one.

I tested a LaTeX file tagged with hidden attribute instead of tw4winExternal and tw4winInternal styles, imported it to DVX, simulated translation and exported. The result is positive, but is yet to be practically tested. I think that flexibility of the taglist may allow DV users to create a tagging sequence which would ensure smooth import and export, tag protection and readability in DV work area.

Other CAT tools compatibility

Unknown. Please, submit your findings.

Known issues

As has been said above, on some local versions you must have a comma "," instead of the semicolon ";" when setting the number of characters to be sought: [^32]{2,}

Fuzzying Wordfast glossary

Quite logically, however surprising to me, I realized that Tortoise Tagger can cope with the task of fuzzying Wordfast glossary. Since the tagger is merely a find/replace batch utility, you can instruct it to run as many passes as you need, and separate all endings with an asterisk from the word stem, like change 'playing' into 'play*ing'.

HowTo

Word's F/R machinery has a very useful feature making it possible for you to instruct it to find anything that is *at the end of the word.* This is done by means of adding the '>' symbol after the string you want to locate at the end of the word, thus, if you wish to separate all 'ing' endings with an asterisk, you should have the following double entry in the taglist:

```
~~~WC-ON
(ing>)   *\1
```

Which means: with wildcards mode ON, find every 'ing' string at the end of the word ((ing>)) and replace it with an asterisk (*) and the same found string (\1).

Again, as with building your own taglist, you should experiment a bit. If the tagger fuzzies words you don't want it to, it's a good idea to make them bold first, and then instruct the tagger to fuzzy only plain text words. Remember, that once you save your glossary as text, all the formatting is lost. Here's a simple theoretical example I made, the meaning of taglist entries is explained by the taglist comments.

```
~~~FindNotBold
~~~WriteBold
%%% the 2 commands above bold everything not already bolded
~~~WC-OFF
%%% literal pass, because a 'wildcarded' one can bring
unexpected results
~~~HWord-ON
%%% finding only whole words, to avoid hits with 'combed',
'remembered'
bed
red
~~~WC-ON
%%% Wildcards mode activated to cover ALL occurrences of '-
ed' ending
(ed>)    *\1
```

Here, again I close my eyes and see the fuzzying taglists for various languages updated and uploaded to 'files' section of the Wordfast group, for other folks to use. If someone actually volunteers to create such a taglist, and again, someone would like to update it, please, bear in mind that you can either insert your lines and comments in the appropriate location of the taglist or add your entire sequence at the bottom, resetting all F/R parameters, unbolding or unhiding the entire glossary and then performing what *you* deem necessary from scratch. I have not tested this opportunity to the extent making it possible for me to make any practical recommendations. Well, seriously, I do believe, that, unlike with TMs and glossaries supposedly freely shared on Wordfast group, this idea is not completely utopian and lunatic, if it is, I hope there are enough lunatics out there. :)

Unfuzzying

Unfuzzying the glossary can be done by hand or using the following taglist:

```
~~~WC-OFF
*
```

(There is a tab after the asterisk, but this is not mandatory.) Once again, keep in mind that all these are just F/R passes, read Word's help, use your logic, play a bit, kick your cat (don't do it, just kidding!) and you will have a working solution. Another thing is to have these two lines at the top of every *fuzzying* taglist, to avoid multiple asterisks in the terms.

Some document tweaking

The commands which deal with highlighting and double strike through font attribute came around when a member of Wordfast list faced a problem when he had a pretranslated Portuguese-English document with improperly set language attributes – entire text was made English. The translator needed to mark the Portuguese text as untranslatable, but could not perform a F/R pass guided by language ID because it was wrong.

Among various responses to his another appeal concerning comparison of documents (God bless Wordfast Yahoo group!) there was a suggestion to use the TM resulting from these documents (AFAIR). This prompted me an idea to edit the TM and use it to set the untranslatable attribute to all source or target segments. The workflow is as follows[3]:

Make a copy of the document. Create an empty TM and clean the document into it. Using Word table or Excel, rip off everything unnecessary and get a column of segments. Save as text.

Edit this one-column document, adding the required command(s) at the top, in our case it could be

```
~~~WriteHilite
Technical specifications of the kukaramba.
... <entire TM>
```

and you would have to set 25% grey manually in Word prior to running the tagger. If you have long dashes or other characters which are stored in Wordfast TM not the way they appear in Word document, you should replace them with a hard return to enable the tagger to format at least most of the segments.

These are general comments on why these have been implemented, some experimenting will definitely produce positive results.

[3] I am speaking about Wordfast TMs here, Trados and DV users will have to go greater lengths to achieve this.

Things I do not understand, but...

Tortoise Tagger
version 1.01

Links

Word

http://word.mvps.org

Latex

http://www.ctan.org

http://www.tug.org/begin.html

VBA

http://www.podmonkeyx.com/codesamples.asp

Please, submit your links which you consider useful. I will post them on the tagger's home page.

Credits

The original idea of tagging Latex in this way belongs to David Daduc, a freelance translator and Wordfast trainer from Prague, wordfast@volny.cz.

Some fundamental VBA knowledge along with critical advice was supplied by Arkady Vysotsky, author of Plus Toyz, nodice@fm.com.ua.

Links to LaTeX files to test the tagger, and a very useful huge file were supplied by Robin Laakso from the TUG office (http://www.tug.org)

Thanks to the members of Wordfast Yahoo group for their advice, support and cheering me up a bit: http://groups.yahoo.com/group/wordfast/

Thanks to the members of DejaVu Yahoo group for their advice and support: http://groups.yahoo.com/group/dejavu-l/

Hooptedoodle

You see, the chances that I get another LaTeX job are next to nothing, the volume of what I've already translated makes me think I've used up my share of LaTeX translation for this life. I could have just as well sit back or play with kids, and so could David when he dug up reference and gave me his advice. Please, follow this line, not only you will enjoy it, but the good you do will definitely return to you some sunny day.

I request folks out there to submit their corrections, notes and taglists to me at ttagger@accurussian.net, and I will keep it updated and expanding. When you submit you list, please, include your comments in the header, including your technical info and your personal and copyright data. The thousands of taglists will be posted as they are received from you.

Do I sound convincing? Well, time will tell...

Thank you and Happy translating!
Aleksandr Okunev
http://www.accurussian.net

In memory of Eduard Rjeutski
who suddenly and unexpectedly died
on December 17, 2004.
God rest his soul.

~~~~~~

Written in December 2004 – January 2005
by Aleksandr Okunev,
a freelance translator

~~~~~~