rtftohtml Users Guide

This document contains directions for using the *rtftohtml* filter.
There are two ways that the rtftohtml filter may be used. You can take existing documents and translate them to HTML, or write new documents explicitly for the World Wide Web. This filter should accomodate both uses.

An Overview of rtftohtml

rtftohtml reads up RTF format documents and translates them to HTML. In processing text, the filter chooses HTML markup based on three characteristics. These are

1) The destination of the text. Example destinations are header, footer, nootnote, picture.
2) The paragraph style. Paragraph styles are user-definable entities, but some are pre-defined by the word processing package. For Microsoft Word (on the Macintosh) examples are "Normal" and "heading 1".
3) The text attributes. Examples of text styles are bold, courier, 12 point.

The filter has built-in rules for dealing with destinations. For paragraph and text styles, the rules for translation are contained in a file called html-trans. By modifying this file, you can train rtftohtml to perform the correct translations for your documents. The most common change that you will need to make is to add your own paragraph styles to html-trans.

rtftohtml should produce reasonable HTML output for most documents. Here is what you can expect:

• Your output should appear in a file called "xx.html" where "xx" or "xx.rtf" was your input file name.
• Bold, italic and underlined text should appear with <b>,<i> and <u> markup
• Courier font text should appear with <tt> markup
• Tables will be formatted using <pre> markup (only plain text is supported in tables.)
• Footnotes will appear in a separate document with hypertext links to them.
• Table of contents, indexes, headers and footers are discarded.
• Table of Contents entries and paragraphs with the style "heading 1..6" will generate a hypertext Table of Contents in a separate file. Each table of contents entry will link to the correct location in the main document.
• All paragraph styles use in your document must appear in the file "html-trans". This allows you to create a mapping from any paragraph style to any HTML markup. There are many pre-defined styles in html-trans, including "heading 1..6". (If a paragraph style is not found, a warning will be generated and the text will be written to the HTML file with no special markup.)
• Each graphic in your file will be written out to a separate file. The filename will be "xxn.ext" where "xx" or "xx.rtf" was your input, "n" is a unique number and "ext" will be either "pict" for Macintosh PICT format graphics or "wmf" for Windows Meta-Files format graphics. The HTML file will create links to these files, using either "<A HREF=" or "<IMG SRC=" links. **SINCE most WWW browsers do not understand "wmf" or**

"pict" format files, the link will be to xxn.gif. This presumes that you will run some **other** filter to translate your graphic files to gif.

- Text that is connected with copy/paste-link constructs will generate hypertext links.

How it works

rtftohtml begins by reading html-trans and the character translation files. The rest of the processing is a loop of reading your RTF file and writing HTML. A high level overview of this loop looks like this:

1) Read the next character. In doing so, the filter also reads all of the RTF markup that specifies the destination, paragraph and text styles of the next character.
2) Process the destination information. Normally, text is destined for the "body" of the document. Sometimes, the text belongs in a header, footnote or footer. The filter discards any text for headers, footers. For a footnote, the filter writes the text at the end of the document and generates a link to it.
3) Process any SPECIAL text styles. The filter compares the text style information to see if it matches any entries in the .TMatch table (in html-trans). If there is a match and the entry is for "_Discard", "_Literal", "_Hot", "_HRef", "_Name" or "_Footnote" then the text will be processed accordingly. For example, "_Discard" text is discarded and "_Name" text will generate an anchor using the text as a name.
4) If the text was not SPECIAL, process the paragraph style. The filter takes the name of the paragraph style and looks it up in the list of paragraph styles in html-trans (in the .PMatch table). If the paragraph style is not found in the table it uses the first entry : "Normal". This entry has a nesting level and the name of the HTML "paragraph"[1] markup to use. Using the HTML paragraph" markup name, the filter (using the .PTag table) knows what tags to generate for the text.
5) If the text was not SPECIAL, process the text styles again. The filter compares the text style information to see if it matches any entries in the .TMatch table (in html-trans). In this step, it is possible to match more than one entry. For each matched entry in the .TMatch table, the filter uses the HTML "text" markup name, the filter (using the .TTag table) knows what tags to generate for the text.

Using this process, the filter can generate any HTML markup for any combination of paragraph style and text style.

What about Graphics?

Graphics are imbedded in RTF in either a binary format or an (ASCII) hex dump of that binary. I have never seen a binary format graphic - I don't think that the filter will process binary correctly. It does handle the hex format of graphics, by converting the hex back into binary and writing the binary to a file. The file extension is chosen by looking at the original type of the graphic. The following list shows the file types and their extensions:

1In HTML, there are tags like <h1>, <ol> which describe the paragraph.s, and tags like <b> and <tt> which describe text. I call these "paragraph" markup and "text" markup respectively.

Macintosh PICT	.pict - also, 256 bytes of nulls are prepended to the graphic. This is to conform to the PICT file format.
Windows Meta-files	.wmf [2]
Windows Bit-map	.bmp

In addition, the filter produces a link to the file containing the graphic. Now, since the above graphic formats are not very portable, the filter assumes that you will convert these files to something more useful, like GIF. So the format of the link is:

```
<a href="basenameN.ext">Click here for a Picture</a>
```

where

- `basename` is the name of the input document (without the .rtf extension)
- `N` is a unique number (starting at 1)
- `ext` is an extension. This defaults to GIF, but can be overidden with the -P command line option.

You can also change the link to an IMG form. If you specify the -I command line option, all links to graphics will be of the form:

```
<IMG src="basenameN.ext">
```

There is one other special case. If a graphic is encountered when the filter is in the process of generating a link, the IMG form of the link is used even without the -I command line option.

Special Processing

In the following discussion of SPECIAL processing, I will assume that rtftohtml has not been customized. If it has, the text styles used to create special effects may be different.

Making a Named Anchor

To make a named anchor, you simply enter the name in the document where you would like the anchor to appear. Then format the text using Outline and Hidden. Be careful in formatting the text that you format ONLY the name - be careful not to format leading and trailing spaces or paragraph marks. As an example, if the text - Named Anchor Example - were formatted with Outline and Hidden, it would produce the HTML output :
<a name="Named Anchor Example"></a>

To change the formatting that produces named anchors, you need to modify the entry in html-trans that specifies "_Name" formatting.

Footnote/Endnote Processing

If your RTF document contains footnotes or endnotes, the filter will place the text of the footnotein aseparate HTML document. At the footnote reference mark, the filter will

2 The filter now generates BEAUTIFUL WMF files. Unfortunately, there are no WMF filters on UNIX systems. Use HiJaak Pro, wmf2bmp or your favorite WMF filter to produce the GIF files required by most WWW browsers.

generate a hypertext link to the text of the footnote. This works with either automatically numbered footnotes[3], or user supplied footnote reference marks[+4]

Discarding Unwanted Text

If you have text that you do not want to appear in the HTML output, simply format the text as Hidden and Plain (that is, no underline, outline...).
If you wish to modify the formatting that discards text, you need to change the entry in html-trans that specifies "_Discard".

Imbedding HTML in a Document

Normally, if your RTF document contained the text "<cite>hello</cite>", the translator would output this as: "&lt;cite&gt;hello&lt;/cite&gt;". This ensures that the text would appear in your HTML output exactly as it appeared in the original RTF document. If, however, you want the <cite></cite> to be interpreted as HTML markup, you must format the tags using Hidden and Shadow. The filter will then send the tags through without translation.
When the rtftohtml filter produces HTML markup, it keeps track of the nesting level of tags to ensure that you don't get something like <b><cite>hello</b></cite> which would be incorrect markup. If you imbed HTML markup in your document, the filter will NOT be aware of it. You must ensure that your markup appears correctly nested.
If you wish to modify the formatting for imbedded HTML, you need to change the entry in html-trans that specifies "_Literal".

Customizing rtftohtml

Some customizations of rtftohtml require a little understanding of how the filter work, others require a lot. All of the customizations involve editing either html-trans or one of the character translation files.

html-trans File Format

In html-trans there are four tables. They are .PTag, .TTag, .PMatch and .TMatch. These tables begin with the name (in column one) and continue until the next table starts. All blank lines and lines beginning with a '#' are discarded. '#' lines are typically used for comments. The tables themselves are composed of records containing a fixed number of fields which are separated by commas. The fields are either strings (which should be quoted) integers or bitmasks.

.PTag Table

Each entry in the .PTag table describes an HTML paragraph markup. The format is:
.PTag

3 Look, there is one now!

4+ There is my mark.

#"name","starttag","endtag","col2mark","tabmark","parmark",allowtext,cannest,DeleteCol1,fold,TocStyl

name	A unique name for this entry. These names are referenced in the .PMatch table.
starttag	This string will be output once at the beginning of any text for this markup.
endtag	This string will be output once at the end of any text for this markup.
col2mark	This string will be output in place of the first tab in every paragraph (used for lists)
parmark	This string will be output in place of each paragraph mark. (usually <br> or <p>)
allowtext	If 0, no text markup will be allowed within this markup. (for example <pre> or <h1> don't format well if they contain additional markup.
cannest	If 1, other paragraph markup will be allowed to nest within this markup. (used for nesting lists)
DeleteCol1	If 1, all text up to the first tab in a paragraph will be deleted. (used to strip out bullets that when going to unordered lists (<ul>).
fold	If 1, the filter will add newlines to the HTML to keep the number of characters in a line to less than 80. For <pre> or <listing> elements, this should be set to 0.
TocStyl	The TOC level. If greater than 0, the filter will create a Table of contents entry for every paragraph using this markup.

Sample .PTag Entries

```
"h1","<h1>\n","</h1>\n","\t","\t","<br>\n",0,0,0,1
```

This is a level 1 heading. The "\n" in the start and end-tag fields forcesa newline in the HTML markup. Since newlines are ignored in HTML (except in <pre>) it's only effect is to make the HTML output more readable. There is no difference between the first tab and any other. They both translate to a tab mark. Paragraph marks generate "<br>" followed by a newline (just for looks). Text markup (like <b>) is not allowed within <h1> text, because we leave that up to the HTML client. No nesting is allowed - (see the discussion on nested styles). No text is deleted. Every paragraph using this markup will also generate a level-1 table of contents entry.

```
"Normal","","\n","\t","\t","<p>\n",1,0,0,0
```

This is the default for normal text. Regular text in HTML has no required start and end-tags. The "\n" in the end-tag field forces a newline in the HTML markup. Since newlines are ignored in HTML (except in <pre>) it's only effect is to make the HTML output more readable. There is no difference between the first tab and any other. They both translate to a tab mark. Paragraph marks generate "<p>" followed by a newline (just for looks). Text markup (like <b>) is allowed within Normal text. No nesting is allowed - (see the discussion on nested styles). No text is deleted.

```
"ul","<ul>\n<li>","</ul>","\t","\t","\n<li>",1,1,0,0
```
This is the entry for unordered lists. This generates a "<ul>\n<li>" at the start of the list and "</ul>/n" at the end. There is no difference between the first tab and any other. They both translate to a tab mark. Paragraph marks generate "<li>" preceded by a newline (just for looks). Text markup (like <b>) is allowed, and this entry may be nested - and it allows others to be nested within it. This allows nested lists. No text is deleted.

```
"ul-d","<ul>\n<li>","</ul>","\t","\t","\n<li>",1,1,1,0
```
This entry is identical to the previous except that the DeleteCol1 field is set to 1. This is used to remove bullets (which really appear in the RTF) because we don't want to see them in the HTML.

.TTag Table

Each entry in the .TTag table describes an HTML text markup. The format is:
.TTag
"name","starttag","endtag"

name	A unique name for this entry. These names are referenced in the .PMatch table.
starttag	This string will be output once at the beginning of any text for this markup.
endtag	This string will be output once at the end of any text for this markup.

Note that unlike the .PTag table, no text markup should appear more than once. (Of course there is no good reason that it should appear.) If you have two entries with <b></b> start and end tags, it would be possible to get HTML of the form <b><b> text</b></b>. I don't know if this is invalid markup, but it sure is ugly.

.PMatch Table

Each entry in the .PMatch correlates a paragraph style name to some entry in the .PTag table. The format is:
.PMatch
"Paragraph Style",nesting_level,"PTagName"

Paragraph Style	The paragraph style name that appears in the RTF input.
nesting_level	The nesting level. This should be zero except for nested list entries.
PTagName	The name of the .PTag entry that should be used for paragraphs with this paragraph style.

Sample .PMatch Entries

```
"heading 1",0,"h1"
```
This is a level 1 heading. Any paragraphs with this paragraph style will be mapped to the entry in the .PTag table named "h1".

```
"numbered list",0,"ol-d"
```
This is used for numbered lists. Any paragraphs with this paragraph style will be mapped to the entry in the .PTag table named "ol-d".

```
"numbered list 2",2,"ol-d"
```
This is an entry for a nested paragraph style. The nesting level of two is used to indicate that this paragraph should appear in the HTML nested within two levels of paragraph markups. The paragraph marked with this style may only appear after a paragraph style that has a nesting level of 1 or greater.

.TMatch Table

Each entry in the .TMatch table describes processing for text styles. The format is:
.TMatch
```
"Font",FontSize,Match,Mask,"TextStyleName"
```

Font	The name of a Font, or "" if all fonts match this entry.
FontSize	The point-size of the font, or 0 if all point sizes match this entry.,
Match	A bit-mask, where each bit represents a text attribute. These bits are compared to the attributes of the style being output. They must match for this entry to be matched. One in a bit position means that the text style is set, a zero is not set.
Mask	A bit-mask, where each bit represents a text attribute. In comparing the style of the text being processed, to the Match bit-mask, this field is used to select the bits that matter. If a zero appears in a bit-position, then that style attribute is ignored (for the purpose of matching this entry.) Only 1 bits are used in the above comparision.
TextStyleName	This is either the name of an entry in the .TTag table indicating the HTML markup to use, or it is one of "_Discard", "_Name", "_HRef", "_Hot", or "_Literal".

```
The order of bits in the Match and Mask bit-maps are:
#     v^bDWUHACSOTIB - Bold
#     v^bDWUHACSOTI - Italic
#     v^bDWUHACSOT - StrikeThrough
#     v^bDWUHACSO - Outline
#     v^bDWUHACS - Shadow
#      v^bDWUHAC - SmallCaps
#      v^bDWUHA - AllCaps
#      v^bDWUH - Hidden
#      v^bDWU - Underline
#      v^bDW - Word Underline
#      v^bD - Dotted Underline
#      v^b - Double Underline
#      v^ - SuperScript
#      v - SubScript
```

Sample .PTag Entries

```
# double-underline/not hidden -> hot text
# double-underline/hidden -> href
#      v^bDWUHACSOTIB,v^bDWUHACSOTIB
"",0,00100000000000,00100010000000,"_Hot"
"",0,00100010000000,00100010000000,"_HRef"
```

The first entry will match any text formatted with double underline EXCEPT if it is hidden text. This is accomplished by using those two bits to compare (the MASK field) and having a 1 in the double underline bit and a zero for the hidden text bit. The second entry will match any text formatted with BOTH double underline and hidden text. Any text that matches the first will be treated as the hot text of a link. Any text that matches the second will be taken as the href itself. (The filter requires that the HRef text immediately precede the Hot text.)

```
# Regular matches - You can have multiple of these active
# monospace fonts -> tt
"Courier",0,00000000000000,00000000000000,"tt"
```

This will match any text that uses the Courier font and mark it using the HTML text markup appearing in the .TTag table with the entry name "tt".

```
# bold -> bold
#      v^bDWUIACSOTIB,v^bDWUIACSOTIB
"",0,00000000000001,00000000000001,"b"
```

This will match any text that has bold attributes and will mark it using the HTML text markup appearing in the .TTag table with the entry name "b". Note that bold text using the Courier font would match both this entry and the previous. This will yeild markup of the form <b><tt>hi</tt><b>. Note that "b" is the name of an entry in the .TTag table, not the HTML markup that is used!

Adding Paragraph Styles

To add a new paragraph style, simply go to the .PMatch table and add an entry to the end. Put the name of the paragraph style (quoted), the nesting level (usually zero) and the name of the .PTag entry that should be used.

Command Line Options

The syntax of the rtftohtml command is as follows:

```
rtftohtml [-i] [ -V] [-o filename] [-P extension] [-T] [-G][file]
```

-i	Indicates that imbedded graphics should be linked into the main document using an IMG tag. The default is to use an HREF style link.
-V	Prints the current Version to stderr.
-o filename	Indicates that the output file name should be "filename". If any other files are created (such as for graphics,) the basename of the other files will be "filename" without ".rtf" if it is present in the name.
-P extension	Use "extension" as the extension for any links to graphics files. The default for this is "gif".
-T	Indicates that no table of contents file is to be generated.
-G	Indicates that no graphics files should be written. The hypertext links to the graphics files will still be generated. This is a performance feature for when you are re-translating a document and the graphics have not changed.
"file"	The file name to be processed. If no file is given, standard input is used. If standard input is used, the body of the document will be written to standard output (unless overridden by the -o option.) If a file name appears, the output is written to "filename" with ".html" as an extension. (If ".rtf" appears as an extension on the original input file, it is stripped before appending the ".html")

Nested Lists

Nested lists can be made from an RTF document by using a different style for each level of indentation. The styles "bullet list 1" "numbered list 2" ... represent different levels of nesting. The only rule for use is that no levels of nesting are skipped. For example, a "ol 3" paragraph must not appear immediately after a "Normal" paragraph. It must follow a paragraph with a nesting level of 2 or higher. For examples of nesting see the sample style sheet. .rtf