



>

Jazz HTML Plugin

>

Version 1.12

◇

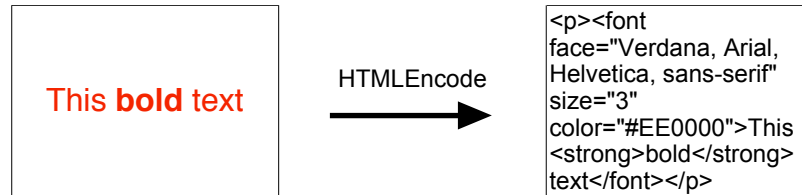
Updated 29 September, 2005

◇

Introduction

◆ Overview

This plugin converts styled text within *FileMaker Pro 7* to HTML-encoded text. For example, a simple paragraph might convert as follows:



Other variations are possible, which include the HTML headers and footers (<html>, <header> and <body> tags, etc), or simple text encoding (ignoring style formatting).

The plugin has been written to allow you flexibility with the tags used. For example, you can choose for bold instead of shown above. Through clever customisation of tags you can obtain limited style-sheet functionality.

◆ Shareware or Freeware

This plugin is shareware. To allow you to test the plugin's suitability for your purpose(s) the main function call of the plugin will function unregistered for fifty operations¹. That is, *HTMLEncode* may be called 50 times before it stops working. After 50 operations the *HTMLEncode* function will cease to work. If you wish to continue with its use, you are obliged to register it at <http://www.jazzmedia.com.au>. All the other functions, in particular *HTMLEncodeRaw* will continue to work thereafter. You may keep the plugin and use this functionality for free. However if you wish to distribute the plugin for any purpose you must register it for distribution.

The basic user license allows the plugin to be used by a single person or installed on a single computer. If you wish to distribute this plugin with your solution there are separate licenses available on a per-product basis. Consult the web-site for more details.

Once you have received a registration serial number, you can activate your serial number using the *HTMLRegister* function, as described below. Single-user licensees may instead use the sample *FileMaker Pro* file to activate their registration.

◆ Disclaimer

The plugin is offered "as-is", and comes with no warranty of any kind. You are responsible for determining its suitability for your particular needs, and to ensure that it performs accordingly. Please do not obtain a license for the plugin until

¹ This count includes use of the sample FileMaker file provided

you have so-determined the plugin's suitability and effectiveness as per your requirements.

Conventions

◆ Definitions

The word 'function' and 'command' are used interchangeably in this document. While technically there is a difference between the two, FileMaker only provides a single method of access so differentiation between the terms is not important.

◆ Calling functions

This extension makes use of the *FileMaker* plugin architecture. Plugin functions can only appear as part of a calculation statement, even if you do not care about the function's returned value. There are two ways to use these functions. You may use which ever suits your scripting style.

The first method is to create an *If-End If* pair of commands, with no script steps in-between. Because the body of the *If-End If* statement is empty, the return value of the function is ignored. The *HTMLSetBoldTag* function could be called this way:

```
If [HTMLSetBoldTag("STRONG")]  
End If
```

A more compact way to achieve the same result is to create a global field called *temp* (for example) which is reserved for offloading unwanted function results. The same *HTMLSetBoldTag* function would now be called like this:

```
Set Field [mytable::temp, HTMLSetBoldTag("STRONG")]
```

The latter is the method I prefer, and is the method used in the examples found throughout this document. In examples where there is no meaningful result returned, the entire *Set Field* encapsulation is omitted for brevity. For example, the above statement might simply be written as:

```
HTMLSetBoldTag("STRONG")
```

However to be implemented, this command needs to be placed into a *FileMaker* script step, such as the *Set Field* step shown immediately preceding it.

Function Summary

◆ HTMLVersion

This function takes no parameters and returns a text string describing the plugin, its registration status and the version number of the plugin. For example:

```
Set Field [mytable::regstatus, HTMLVersion]
```

will place a string along the lines of "Jazz Media HTML 1.00 - Registered" into the field *regstatus*. The *TextToNum* function can be used to return the version number on its own (as a decimal number), from which comparisons can be made. (For this reason the version number will contain only a single decimal point, eg. "1.21" and not "1.2.1", as has become common practise today). The following example places the numeric version in a field called *plugin version*.

```
Set Field [mytable::plugin version, TextToNum(HTMLVersion)]
```

◆ HTMLRegister

This function is used to activate plugin registration. The serial number provided to registered users should be supplied to this function. For example, if the serial number supplied is "12345" you can activate your registration with the following command:

```
HTMLRegister(12345)
```

Single user licenses save the registration details in a preference file, so the registration only needs to be activated once. However a registration code purchased for product distribution does not get saved to a preference file. Rather this function should be placed in a start-up script, to be called every time the runtime application is started.

For runtime solutions the registration code need only be supplied once to register use for all open databases. However it is recommended that a script containing the registration code is placed in the start-up script for every database file which uses this plugin (see *FileMaker's* document preferences).

If you accidentally supply an invalid serial number, you will need to quit and re-launch FileMaker before the correct serial number will be accepted. You can check your registration status at any time by using the *HTMLVersion* function, described above.

◆ HTML Encode

This function constitutes the core functionality of the plugin. It converts styled text (formatted text) into equivalent HTML code. The exact details of this encoding can be configured using the *HTMLSet...* series of commands. Consult the individual commands to understand the affect they gave on *HTML Encode*.

The *HTML Encode* command takes a single parameter which is *FileMaker* text, usually straight from a field. This command returns the equivalent HTML code, excluding `<html>`, `<header>` and `<body>` tags. Such tags may be added by using the *HTMLHeader* and *HTMLFooter* commands. Separation of the header and footer codes from the content is intentional, as it allows you to build quite complex HTML pages, built from multiple *FileMaker* fields.

The *HTML Encode* command can be called as follows:

```
Set Field [mytable::html, HTML Encode(mytable::text)]
```

This places converts the styled text from the field called *text*, and places the resulting HTML code into the field called *html*. To create a complete HTML page built from this single field you could use:

```
Set Field [mytable::html, HTMLHeader("My page title", "en") &  
HTML Encode(mytable::text) & HTMLFooter]
```

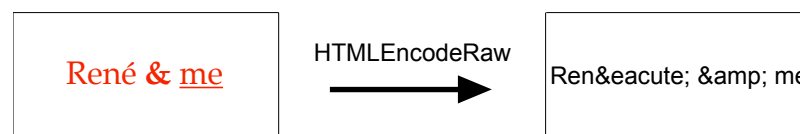
◆ HTML Encode Raw

This function encodes raw text into equivalent HTML, ignoring any styling which may be present. It replaces any return characters into a break tag (eg. `
`), but otherwise no other tags are included in the output. Two examples of use follow:

```
HTML Encode Raw("René & me")
```

```
HTML Encode Raw(formattedTextField) [where the text field contains "René & me"]
```

These statements perform the following translation:



Anticipated use for this function is composing HTML where a single *FileMaker* field makes up only a portion of an HTML paragraph, thus allowing users to merge several fields as they see fit, adding paragraph and other formatting manually. This command could also be used to ignore rogue formatting.

◆ HTML Set Paragraph Format

When using the *HTML Encode* command, paragraphs can be interpreted in different ways. In some situations a single return character should generate a new paragraph. If you are listing items beneath each other you may not want them to be considered as separate paragraphs. Instead you may choose that two consecutive returns (ie. a single blank line) should become a paragraph, and

single occurrences should become line breaks (the break tag, or
). There may instead be cases where you do not want any paragraphs to be created, and all returns characters should be encoded as breaks.

This command allows you to chose which behaviour is appropriate for your situation. The command takes a single numeric parameter: 0, 1 or 2, representing each of the behaviours described above:

<i>Parameter</i>	<i>Meaning</i>
2	Two consecutive returns indicate a paragraph
1	A single return indicates a new paragraph
0	Do not form any paragraphs

In the first case (2) remaining (single) return characters will be encoded as breaks. An example of use follows:

```
HTMLSetParagraphFormat(2)
```

The paragraph format is reset to the first option (2) by the *HTMLResetAll* command.

◆ HTMLSetParagraphTag

This function defines the tag used for a paragraph. It defaults to “p” which causes <p> and </p> to be used as paragraph open and close tags. If you set it to the empty string “” then no paragraph tags will be included anywhere.

You may experiment with including parameters as part of this tag. For example, if you provide *p class="myclass"* to this function, the opening tag will be <p class="myclass">. The closing tag ignores everything after the space, and so will still be </p>.

It is also permissible to set paragraphs to use any custom text you choose. The tag contents is reset to “p” by the *HTMLResetTags* and *HTMLResetAll* commands.

◆ HTMLSetBreakTag

This function defines the tag used for a line break. It defaults to “br” which causes
 to be used as the line break tags. f you set it to the empty string “” then no break tags will be included anywhere.

You may experiment with including parameters as part of this tag. For example, if you provide *br clear="all"* to this function, then all breaks will be represented by the tag <br clear="all">. You may also set it to “br/” to use XHTML compliant tags
. There is no closing tag for a line break.

It is also permissible to set break tags to use any custom text you choose. The tag contents is reset to “br” by the *HTMLResetTags* and *HTMLResetAll* commands.

- ◆ **HTMLSetBoldTag**
HTMLSetItalicTag
HTMLSetUnderlineTag

- HTMLSetSubscriptTag**
HTMLSetSuperscriptTag

These functions define the tags used for bold, italic, underline, subscript and superscript text. They default to “*strong*”, “*em*”, “*u*”, “*sub*” and “*sup*” respectively, causing and to be used as the bold open and close tags, and so on. Providing the empty string (“”) to any command will disable use of the respective tag entirely.

You may use these functions to choose the old-style and <i> tags instead of and if you desire. You may also experiment with including parameters as part of these tag. For example, if you provide *b class="myclass"* to *HTMLSetBoldTag*, then opening bold tags will appear as <b class="myclass">. Closing tags are based on anything before the first space, and so would be in this case, as expected.

It is also permissible to supply other custom text to these functions. The tag contents are reset to the listed default values (above) by the *HTMLResetTags* and *HTMLResetAll* commands.

- ◆ **HTMLSetFontSizes**

This function defines the font point sizes used for HTML sizes 1 through 7. Calling the function as follows:

```
HTMLSetFontSizes(10; 12; 14; 18; 24; 32; 48)
```

configures the *HTMLEncode* command so that font sizes up to and including 10 pt will be represented as HTML font size=1. Font sizes up to 12 pt (and >10 pt) will be represented by HTML font size=2, and so on. Finally font sizes up to 48 pt² (and over 32 pt) are represented by HTML font size=7. The parameters provided should always be in ascending order.

Font size mappings are reset to their default values by the *HTMLResetFontSizes* command. The default values are identical to those used in this example.

- ◆ **HTMLSetFontFace**

The *HTMLEncode* command does not recognise font families (*Times*, *Helvetica*, etc) used in *FileMaker*. This is due to a limitation (or perhaps a lack of documentation) in the *FileMaker* plugin API³. As a result the font family (face) can not be automatically ascribed to tags. To best overcome this issue, the user can manually specify the font which the HTML will be created with. For example:

```
HTMLSetFontFace("Times, Times New Roman, serif")
```

tells the plugin to assume all text is *Times*, defaulting to a serif font if the HTML browser can not locate this font.

² The astute will realise that his command does not specify how to treat font sizes over 48 point. They must of course be the largest size which is 7. This happens to be the same size as for fonts over 32 point but up to 48 point. As a consequence the last parameter for this function (48) actually has no affect on the outcome whatsoever. The last parameter must be included however, simply because its omission would be more confusing to most than not.

³ Application programming interface: the interface through which plugins communicate with the FileMaker application.

If you do not specify a font family (face), all fonts will default to *Verdana, Arial, Helvetica, sans-serif*. You may call this function with an empty string to stop any font family from being provided in the `` tag. The function *HTMLResetAll* sets the font family back to the default setting.

◆ HTMLSetFontFlags

This function allows you to configure options for how and when the `` tag is used. At present it allows you to omit the font tag or certain parameters within the tag in different circumstances, however this functionality may be expanded in the future.

The command takes a single numeric parameter, which is a sum of values representing each option. The options and values available are shown in the first five rows of the table which follows. By default all these options are off.

Option	Value
Omit <i>color</i> parameter when text is black	1
Omit entire <code></code> tag when no contents*	2
Underline	4
Outline	8
Shadow	16
Don't omit anything	0
Omit entire <code></code> tag always	255

For example, to omit the *color* parameter when text is black, and to omit the entire `` tag when there is no contents, you add the values for these two options together and make the call:

```
HTMLSetFontFlags( 1+2 )
```

or simply

```
HTMLSetFontFlags( 3 )
```

To reset to the default behaviour, call

```
HTMLSetFontFlags( 0 )
```

To omit the entire `` tag you could simply add $4 + 8 + 16$ and use this value (28), however to ensure compatibility with possible future options it is recommended that you use a value of 255 instead:

```
HTMLSetFontFlags( 255 )
```

* The *omit when no contents* option affects empty paragraphs, which might be used for extra vertical spacing between text. Although enabling this option makes much neater code, it may affect the line height of the empty line produced. For this reason it is not the default setting.

◆ HTMLHeader

This function returns a standard HTML header, which includes the opening `<html>`

tag, a full `<head>` tag, and an opening `<body>` tag. The command can take two parameters, specifying the web-page title, and the two-letter language. For example, the following command:

```
HTMLTitle("Fred's Home", "en")
```

results in the following HTML code:

```
<html lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
  <title>Fred's Home</title>
</head>
<body>
```

By placing *HTMLHeader* and *HTMLFooter* statements either side of one or more calls to *HTMLEncode* the user can quickly build custom HTML pages. Note that the *HTMLHeader* command is designed to facilitate quick simple HTML pages. It is not designed to comprehensively cover all options. For example, it does not include a *DOCTYPE* tag. More complex requirements would see the user creating his or her own custom header and footer html code.

◆ HTMLFooter

This function is designed to complement the *HTMLHeader* command, and facilitates quick HTML page creation. It takes no arguments, and generates the following HTML code:

```
</body>
</html>
```

See the *HTMLHeader* functional description for an explanation of typical use.

◆ HTMLResetFontSizes

This function resets the font sizes (which were set by *HTMLSetFontSizes*) to their default values. See *HTMLSetFontSizes* for further explanation. A call to *HTMLResetFontSizes* is equivalent to the following command:

```
HTMLSetFontSizes(10; 12; 14; 18; 24; 32; 48)
```

◆ HTMLResetTags

This function resets all the tags to their predetermined default values. It affects the paragraph, break, bold, italic, underline, subscript and superscript tags. The tags are set to the following defaults respectively (only opening tags are shown):

```
<p>, <br>, <strong>, <em>, <u>, <sub>, <sup>
```

◆ HTMLResetAll

This function resets all parameters to their default values. It resets all of the following:

Functionality...	Equivalent to command...
reset all font sizes	<i>HTMLResetFontSizes</i>
reset all tags	<i>HTMLResetTags</i>
reset paragraph format	<i>HTMLSetParagraphFormat(2)</i>
reset HTML font face	<i>HTMLSetFontFace("Verdana, Arial, Helvetica, sans-serif")</i>

See the descriptions of the equivalent commands for the details of the actual values obtained. These values are identical to the defaults configured when *FileMaker Pro* is launched.

Supported styles

While *HTML Encode* tries to retain as much styled information as it can, there are limits to what it can transfer. The following is a summary of the styles supported:

Style	Support
Bold	√
Italic	√
Underline	√
Outline	x
Shadow	x
Condensed, Extended	x
Strikethrough	x
SmallCaps	x
Superscript, Subscript	√
Forced Uppercase, Lowercase, Titlecase	x
Word underline, Double underline, etc.	x
Text size	√
Font family	manual
Text colour	√
Font ascent, descent	x

At a glance, the support looks quite thin. Actually many of the unsupported styles are due to limitations of standard HTML. HTML simply does not support most of the styles listed in the table. It does support the most common ones however.

A key point to note is that the font family (eg. *Times*, *Helvetica*) information is only supported through manual specification. Although this information *is* present in *FileMaker Pro 7*, *Jazz Media* has been unable to determine a method to extract this information. We hope to rectify the situation if and when the information becomes available.

Although the font family is not available, we expect that in many situations this will not be an issue. This is because for database content the font family may not be an important part of the information. Alternatively, perhaps the font family will already be known and it can be reapplied separately. The *HTMLSetFontFace* function allows you to tell the plugin what font family the HTML code should be created in.

Supported character encodings

The following table summarises the character encodings (or replacements) which take place during an *HTMLEncode* (or *HTMLEncodeRaw*) operation:

"	"	•	•	–	–
&	&	¶	¶	—	—
<	<	β	ß	“	“
>	>	®	®	”	”
`	«	©	©	‘	‘
Ä	Ä	™	™	’	’
Å	Å	´	»	÷	÷
Ç	Ç	¨	¨	◊	◊
É	É	≠	≠	ÿ	ÿ
Ñ	Ñ	Æ	Æ	ÿ	Ÿ
Ö	Ö	Ø	Ø	/	⁄
Ü	Ü	∞	∞	€	€
á	á	±	±	‹	‹
à	à	≤	≤	›	›
â	â	≥	≥	fi	fi
ä	ä	¥	¥	fl	fl
ã	ã	μ	µ	‡	‡
å	å	∂	∂	·	·
ç	ç	Σ	∑	,	‚
é	é	π	∏	„	„
è	è	π	π	‰	‰
ê	ê	∫	∫	Â	Â
ë	ë	^a	ª	Ê	Ê
í	í	°	º	Á	&Acute;
ì	ì	Ω	Ω	Ë	Ë
î	î	æ	æ	È	È
ï	ï	ø	ø	Í	Í
ñ	ñ	¿	¿	Î	Î
ó	ó	¡	¡	Ï	Ï
ò	ò	¬	¬	Ì	Ì
ô	ô	√	√	Ó	Ó
ö	ö	f	ƒ	Ô	Ô
õ	õ	≈	≈	Ò	Ò
ú	ú	«	«	Ú	Ú
ù	ù	»	»	Û	Û
û	û	...	…	Û	Ù
ü	ü	[]*	 	^	ˆ
†	†	À	À	˜	˜
°	°	Ã	Ã	-	¯
¢	¢	Õ	Õ	,	¸
£	£	Œ	Œ		
§	§	œ	œ		

* non-breaking space

Appendices

◆ Limitations

Other than some style limitations (see *Supported Styles*, above), the only known limitations relate to *Unicode* text. The plugin is *not* unicode-aware.

Special characters, such as those with accents, Greek characters, and other symbols are only converted into HTML codes if they have a representation in the standard 8-bit Roman character set of the platform. For example, the lower case Greek letter *delta* (δ) is not available in the *Windows* Roman character set, but is available on the *Macintosh*. Therefore this character does not convert to HTML on *Windows*. There are examples the other way too. On the whole, such characters are used very rarely and are unlikely to be an issue for most users.

◆ Version History

- 1.00 Released to the public (September 2004)
- 1.10 Allowed `
` to occur within a font tag, creating more efficient HTML. Added *SetFontFlags* function. (January 2005)
- 1.11 Corrected `–` and `—` encoding issue. Increased trial limit to 50 operations. (March 2005)
- 1.12 Fixed an issue where closing HTML tags weren't applied at the end of text, under certain conditions when the font tag (size and colour) was disabled. (September 2005)

◆ About Jazz Media

Jazz Media is an Australian-based company, who's main business is in creating Multimedia products for clients. It specialises in CD-ROMs and dynamic web sites requiring a database back-end. Use of an in-house database engine for CD-ROMs provides extremely fast access to data, and means that clients can distribute solutions royalty free.

While creating major client projects, it has been necessary or cost-effective to develop a series of plugins for *FileMaker Pro* — either to solve specific developmental problems, or to add functionality to programs and bundled as part of the commercial products. After years of successful operation, many of these plugins have been repackaged and released to the public as shareware or freeware.

If you are in the market for standalone or web-based products with more than a few pages of information, please email john@jazzmedia.com.au. You can view many completed projects on the web site at <http://www.jazzmedia.com.au/products.html>.