From EULER Project

PmWiki: Custom Markup

Introduction

PmWiki's markup translation engine is handled by a set of rules; each rule searches for a specific pattern in the markup text and replaces it with some replacement text. Internally, this is accomplished by using PHP's "preg_replace(approve sites)" function.

Rules are added to the translation engine via PmWiki's Markup() function, which looks like

```
Markup($name, $when, $pattern, $replace);
```
where `$name` is a unique name (a string) given to the rule, `$when` says when the rule should be applied relative to other rules, `$pattern` is the pattern to be searched for in the markup text, and `$replace` is what the pattern should be replaced with.

For example, here's the code that creates the rule for `''emphasized text''` (in *scripts/stdmarkup.php*):

```
Markup("em", "inline", "/''(.*?)''/", "<em>$1</em>");
```
Basically this statement says to create a rule called "em" to be performed with the other "inline" markups, and the rule replaces any text inside two pairs of single quotes with the same text ($1) surrounded by `<em>` and `</em>`.

The first two parameters to Markup() are used to specify the sequence in which rules should be applied. The first parameter provides a name for a rule -- "`em`" in the example above. We could've chosen other names such as "`''`", or even "`twosinglequotes`". In general PmWiki uses the markup itself to name the rule (i.e., PmWiki uses "`''`" instead of "`em`"), but to keep this example easier to read later on we'll use a mnemonic name for now.

The second parameter says that this rule is to be done along with the other "inline" markups. PmWiki divides the translation process into a number of phases:

```
_begin      start of translation
fulltext    translations to be performed on the full text
split       conversion of the full markup text into lines to be processed
directives  directive processing
inline      inline markups
links       conversion of [[links]], url-links, and WikiWords
block       block markups
style       style handling
_end        end of translation
```

This argument is normally specified as a left-angle bracket ("before") or a right-angle bracket ("after") followed by the name of another rule.

Thus, specifying "inline" for the second parameter says that this rule should be applied when the other "inline" rules are being performed. If we want a rule to be performed with the directives -- i.e., before inline rules are processed, we would specify "directives" or "<inline" for the second parameter.

A significant rule in terms of ordering is "" which substitutes variables -- if you say "<" then your markup will be processed before variables are substituted whereas if you say ">" then your markup will be processed after variables are substituted.
The third parameter is a Perl-compatible regular expression. Basically, it is a slash, a regular expression(approve sites), another slash, and a set of optional modifiers(approve sites).

The example uses the pattern string `"/''(.*?)''/"`, which uses `''(.*?)''` as the regular expression and no options. (The regular expression says "find two single quotes in succession, then as few arbitrary characters as are needed to make the match find something, then two additional single quotes in succession"; the parentheses "capture" a part of the wikitext for later use.)

The fourth parameter is the replacement text that should be inserted instead of the marked-up wikitext. You can use `$1`, `$2`, etc. to insert the text from the first, second etc. parenthesised part of the regular expression.

In the example, we have `"<em>$1</em>"`, which is an `<em>`, the text matched by the first parentheses (i.e. by the `.*?` section of the pattern), and `</em>`.

Here's a rule for `@@monospaced@@` text:

```
Markup("@@", "inline", "/@@(.*?)@@/", "<code>$1</code>");
```
and for a `[:comment ...:]` directive that is simply removed from the output:

```
Markup("comment", "directives", "/\\[:comment .*?:\\]/", '');
```
Okay, now how about the rule for `'''strong emphasis'''`? We have to be a bit careful here, because although this translation should be performed along with other inline markup, we also have to make sure that the rule for `'''` is handled *before* the rule for `''`, because `'''` also contains `''`. The second parameter to Markup() can be used to specify the new rule's relationship to any other rule:

```
Markup("strong", "<em", "/'''(.*?)'''/", "<strong>$1</strong>");
```
This creates a rule called "strong", and the second parameter "<em" says to be sure that this rule is processed before the "em" rule we defined above. If we wanted to do something after the "em" rule, we would use ">em" instead. Thus, it's possible to add rules at any point in PmWiki's markup translation process in an extensible manner. (In fact, the "inline", "block", "directives", etc., phases above are just placeholder rules used to provide an overall sequence for other rules. Thus one can use "<inline" to specify rules that should be handled before any other inline rules.)

If you want to disable available markup just call e.g.:

```
DisableMarkup("strong")
```
PmWiki's default markup rules are defined in the *scripts/stdmarkup.php* file. To see the entire translation table as the program is running, the scripts/diag.php module adds `?action=ruleset`, which displays the set of defined markup rules in the sequence in which they will be processed. You can see it at CustomMarkup?action=ruleset. You must first enable the action by setting `$EnableDiag` = 1 in your configuration file.

Other common examples

Define a custom markup to produce a specific HTML or Javascript sequence

Suppose an admin wants to have a simple "`(:example:)`" markup that will always produce a fixed HTML string in the output, such as for a webring, Google AdSense display, or Javascript. The Markup() call to do this would be:

```
Markup('example', 'directives',
  '/\\(:example:\\)/',
  Keep("<div class='example'><p>Here is a
    <a target='_blank' href='http://www.example.com'>link</a> to
    <em>example.com</em></p></div>") );
```

- The first argument is a unique name for the markup ("example").
- The second argument says to perform this markup along with other directives.
- The third argument is the pattern to look for "(:example:)".
- The fourth argument is the HTML that "(:example:)" is to be replaced with. We use the Keep() function here to prevent the output from being further processed by PmWiki's markup rule -- in the above example, we don't want the http://www.example.com(approve sites) url to be again converted to a link.

Define a markup to call a custom function that returns content

An 'e' option on the `$pattern` parameter will cause the `$replace` parameter to be treated as a PHP expression to be evaluated instead of replacement text. Thus, a markup to produce a random number between 1 and 100 might look like:

```
Markup('random', 'directives',
  '/\\(:random:\\)/e',
  "rand(1, 100)");
```

This calls the PHP built-in rand() function and substitutes the directive with the result. Any function can be called, including functions defined in a local customization file.

Arguments can also be passed by using regular expression capturing parentheses, thus

```
Markup('randomargs', 'directives',
  '/\\(:random (\\d+) (\\d+):\\)/e',
  "rand('$1', '$2')");
```

will cause the markup `(:random 50 100:)` to generate a random number between 50 and 100.

Note: Be very careful with the /e modifier in regular expressions; malicious authors may be able to pass strings that cause arbitrary and undesirable PHP functions to be executed.
For a PmWiki function to help with parsing arbitrary sequences of arguments and key=value pairs, see Cookbook:ParseArgs.

How can I embed JavaScript into a page's output?

There are several ways to do this. The Cookbook:JavaScript recipe describes a simple means for embedding static JavaScript into web pages using custom markup. For editing JavaScript directly in wiki pages (which can pose various security risks), see the JavaScript-Editable recipe. For JavaScript that is to appear in headers or footers of pages, the skin template can be modified directly, or <script> statements can be inserted using

the $HTMLHeaderFmt array.

How would I create a markup (*(:nodiscussion:)*) that will set a page variable (*{$HideDiscussion}*) which can be used by *(:if enabled HideDiscussion:)* in .PageActions?

Add the following section of code to your config.php

```
SDV($HideDiscussion, 0);        #define var name
Markup('hideDiscussion', '<{$var}',
 '/\\(:nodiscussion:\\)/e', 'setHideDiscussion(true)');
function setHideDiscussion($val) {
  global $HideDiscussion;
  $HideDiscussion = $val;
}
```

This will enable the `(:if enabled HideDiscussion:)` markup to be used. If you want to print the current value of {$HideDiscussion} (for testing purposes) on the page, you'll also need to add the line:
`$FmtPV['$HideDiscussion'] = '$GLOBALS["HideDiscussion"]';`

PmWiki only supports tool tips for external links, can I use custom markup to add tool tips to internal links?

Yes, add the following custom markup to your config.php:
Markup('%title%', 'inline', '/%title%(.*?)"(.*?)"(.*?)%%/', '$1$3'); # Add tool tips to internal links, Example: %title%[[link"tool tip"]]%%

Use the markup with internal links such as:
%title%[[CookBook "cool" | Example]]%%

See also Cookbook:LinkTitles.

It appears that (.*?) does not match newlines in these functions, making the above example inoperable if the text to be wrappen in contains new lines.

If you include the "s" modifier on the regular expression then the dot (.) will match newlines. Thus your regular expression will be "/STUFF(.*?)/s". That s at the very end is what you are looking for. If you start getting into multi-line regexes you may be forced to look at the m option as well - let's anchors (^ and $) match not begin/end of strings but also begin/end of lines (i.e., right before/after a newline).

How do I get started writing recipes and creating my own custom markup?

(alternate) Introduction to custom markup for Beginners