Mihalism Template Engine Guide

What you need to know before editing.

Since June of 2008 Mihalism Multi Host has had a template engine built into its core in order to make editing of the overall design easier for experts and novelists alike. The template engine uses basic HTML syntax to make it easy to understand and modify.

This guide will present you with information on the structure of templates, how they are organized, how they are stored, and much more.

Table of Contents:

Note 1: *This guide assumes the reader has a basic understanding of HTML and PHP.*
Note 2: *This guide is used as a beginners guide, it is not a complete representation.*

This PDF document was created with Mac OS X Leopard. If it does not render correctly for your operating system, then please let us know at www.mihalism.net.

Template Structure

What they look like.

Before you can begin editing and creating your own template files, you must first understand what exactly a template is. Mihalism Multi Host defines a template as a file containing one or more pieces of the overall design of your website. These pieces are separated from the actual core of Mihalism Multi Host so you do not have to filter through a lot of confusing code to find a few snippets of HTML.

All template files for Mihalism Multi Host are stored in the folder ./source/public_html and are defined by the .tpl file extension. All templates must be placed in this folder.

Now that you understand what a template file is, let us take a look at a basic one:

```
<div class="text_effects">
        <strong>Text to output</strong>
</div>
```

As you can see, templates are nothing more than raw HTML at their most basic level.

In Mihalism Multi Host there are two types of template files. The most common is the basic kind as seen above. The second type are multi-layered template files. Multi-layered template files take several basic templates and place them into a single file in order to save space and not create a lot of small files.

In multi-layered template files, each template contained within the template file is defined by the `<template></template>` tags. Each `<template>` tag has an identifier which is unique to the file that it is placed in and is what Mihalism Multi Host will use to pull only the necessary template from the file.

Here is an example of a multi-layered template file:

```
<template id="text_box_one">
        [some html]
</template>
<template id=”another_text_box">
        [more html]
</template>
```

So now if Mihalism Multi Host just wants to get template “`another_text_box`” from the multi-layered template file, it just has to look for the `<template>` tag with that identifier instead of scanning the entire file.

Variables

How data is transfered.

In template files there are special peaces of code that are known as variables. They look like `<# SOME_STRING #>` or `<# SOME_STRING_2 #>`. Basically they are place holders for a value that will be dynamically generated by the core of Mihalism Multi Host. They can represent anything that Mihalism Multi Host wants to output. They can even be replaced by the contents of another template file if required.

Variables must match the `<# VARIABLE_NAME #>` format to be processed. The names are not required to be capitalized, but making them capitalized does make them easier to spot while skimming the contents of a large template file. Variables must also not span across more than one line or they will not be replaced.

In Mihalism Multi Host variables must be set before the actual template that they are in is parsed. In order to set a variable or multiple variables PHP is required. Here is an example of what two variables being set would look like in the core:

```
$mmhclass->templ->templ_vars[] = array(
        "VARIABLE_NAME" => "Value of variable",
        "VARIABLE_2" => "Another variable value",
);
```

As you can see from above, variable declarations are nothing more than an array with the variable name set as a key and the actual value of the variable set as the value of that key. Variable values can be strings, numbers, PHP $variables, etc.

Controlling Output

Only show what is needed.

Similar to the *if* control structure of PHP, Mihalism Multi Host also has a system that can be used to determine what to output and what not to output. In Mihalism Multi Host PHP if control structures are controlled by the `<if>`, `<elseif>`, `<else>`, and `<endif>` tags. The following are a few examples of how these tags can be put to use.

```
<if="$mmhclass->info->version > 4">
        You are running Mihalism Multi Host v5.x
</endif>
```

```
<if="$mmhclass->info->version > 4">
        You are running Mihalism Multi Host v5.x
<else>
        You are not running Mihalism Multi Host v5.x
</endif>
```

```
<if="$mmhclass->info->version > 4">
        You are running Mihalism Multi Host v5.x
<elseif="$mmhclass->info->version == '4.0.0'">
        You are running Mihalism Multi Host 4.0.0
<else>
        You are not running Mihalism Multi Host v4.0.0 or v5.x
</endif>
```

Note 1: *The `$mmhclass` variable is always passed to template files and it is always available in control structures such as the ones shown above.*

Note 2: *Mihalism Multi Host is not restricted to* `<if>`, `<elseif>`, `<else>`, and `<endif>` tags. There are several other undocumented control structures, but `<if>` is the most common.

Outputting Templates

How to parse templates.

Mihalism Multi Host has two easy to use core functions for outputting templates. Each function parses templates the same way, but how they display the result is quite different.

The function: `$mmhclass->templ->output($filename, $template)`

`$filename` is optional and is the filename of the template file.
No path or file extension is required for the filename.
`$template` is optional and is the template identifier to pick from.

The `output()` function does a forced output of the requested template file. This means that it takes the HTML that has been parsed and places it between the global page header and page footer. Once placed within the global templates, it then outputs the result to the browser.

Note 1: *If `$filename` is not set, then `$mmhclass->templ->html` must contain the HTML to parse.*

The function: `$mmhclass->templ->parse_template($filename, $template)`

`$filename` is optional and is the filename of the template file.
No path or file extension is required for the filename.
`$template` is optional and is the template identifier to pick from.

Unlike the `output()` function, the `parse_template()` function returns the result of a template parsing. It does not force output and does not place it within the global page header and footer.

Note 1: *All template activity is handled by the `$mmhclass->templ` class.*