InScript

A WordPress plugin on steroids.

John Godley © 2005 (http://www.urbangiraffe.com)

INTRODUCTION

InScript is an extensible scripting framework that provides the capability to insert and modify data without needing to modify any WordPress files.

At the simplest level, it is a generic pattern matcher – it searches for specific patterns, or tags, and replaces them with something else. However, instead of replacing these patterns with static text, you can replace them with variables, functions, and even PHP code.

Some of the features that InScript provides are:

- Dynamic variables, which can be used in many places and are automatically updated
- Disable WordPress texturize functions across the whole blog, individual posts, or even individual words
- Conversely, enable texturize, textile, markdown, or any formatting on individual posts or words
- Embed well-defined scripts inside posts and any other part of WordPress, without modifying the theme

The embedded scripts are very powerful, and allow you to do things like:

- Insert post & author information
- Add HTTP meta-values and make them post-specific
- Customise the appearance of words, paragraphs, or posts
- Change date formats on individual sections
- Insert highlighted code
- Insert custom PHP code
- Add custom stylesheets for specific posts

Because of the extensible nature of the plugin, you can add 'scriptlets' (mini-plugins) that provide extra features and yet use the same InScript framework.

NO REALLY, WHAT DOES IT DO?

The best way to explain what InScript does is by providing examples.

Example 1 – Code highlighting

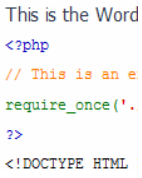
Often you want to highlight some code within a post. This creates several problems:

- Code uses special characters that interfere with HTML
- WordPress mangles a lot of the characters
- It's a chore to color the code by hand

We can insert an InScript tag to do all this for us.

```
This is the WordPress code:
%%format_highlight [file=/wp.php] [wp=off]%%
```

And this produces:



Example 2 - Automatic author information

Still not convinced? Ok, say you wanted to refer to the website of one of your registered authors. Not a problem – just insert their website URL and all is well. But wouldn't it be better to use the information stored within WordPress? Here it is:

```
You can see <a href="%%author_url [login=admin]%%">%%author_fullname [login=admin]%%</a> has updated
his website.
```

Now the information updates automatically whenever the user changes their profile.

Example 3 – Modify times

You want more? In addition to embedding functions within content, you can also apply a function to a whole WordPress element. That is, you can apply it to something such as the date or title of a post. You can choose to do this for all posts, or individual posts.

Here the date of one post changes by giving it a custom field:

```
inscript_the_time = my home lappy, %%time_since [time=%2]%%
```

And now the date for that post (and no other post) changes from this:

Posted on May 26th, 2005

To this:

Posted on my home lappy, 9 days 14 hours ago

Example 4 – Blog title & description

Want to change your blog title for a post? This requires two custom fields:

```
inscript_option_blogdescription = Special offer - buy one stapler for the price of two!
inscript_option_blogname = SuburbanGiraffe
```

Now when you view the blog, the title is as normal:

But when you view the post with the custom field, the title becomes:

Of course, all of these are simplistic examples. Separate plugins already exist to achieve many of the functions described here. The point is that InScript gives you the ability to perform these functions anywhere in your blog, using just one syntax and one plugin. Additionally, you can customise the output exactly as you want, using a wider range of functions. You can cook up all sorts of madness – now, who's for colour-highlighted code in their title?

INSTALLATION

The following steps are required for installation:

1) Download the inscript.zip file
2) Unpack the zip
3) Upload the files and directories to the `wp-content/plugins` directory of your WordPress blog
4) Enable the plugin from the WordPress administration page

INSTALLING SCRIPTLETS

InScript can be extended with scriptlets. These small pieces of PHP code add extra functions to the existing InScript framework.

To install a scriptlet simply upload the file to the `wp-contents/plugins/inscript/` directory of your WordPress blog.

USING INSCRIPT

InScript is based around replacing tags with content. For example, you insert a tag into your post and InScript replaces this tag with the result of some script.

TAGS

InScript has been designed to be fairly flexible when it comes to tags, and this is to make them easy to use yet providing scope for advanced usage. There are three types of InScript tags:

- Variables - `!!variablename!!`
- Short function - `%%function [param1=this] [param2=and that]%%`
- Full function - `<inscript func="function" param1="this" param2="that"/>`

We will look at each of these in turn.

Variable tag

An InScript variable is a piece of data that has been assigned name. This data can be anything at all – text, HTML, even other InScript variables and tags. A variable is defined in the InScript options page, and it's value can be changed at anytime.

Variables are inserted into text by surrounding the variable name with double exclamation marks:

```
!!variablename!!
```

When InScript finds this tag it will replace it with the contents of the associated data. So, if you created a variable called 'address' and set the value to be '15 Big Street, London', then you could embed the variable as follows:

```
If you need to contact me, my address is !!address!!.
```

Each time WordPress displays that section of text, InScript will replace the tag with the current value of the variable.

```
If you need to contact me, my address is 15 Big Street, London.
```

At a later date you might change address. You can update the variable value from the InScript options page and now when WordPress displays the text it will also be updated:

```
If you need to contact me, my address is 18 Little Street, London.
```

Short function tag

A short function tag is a shorthand way of using a full function tag. They are designed to keep typing to a minimum, and also to allow functions to be inserted into parameters of full functions.

A short function is enclosed within double percent characters:

```
%%function [param1=some data] [param2=other data]%%
```

The first word after the double-percent is the function name. This must be an existing InScript function name (detailed later in this guide). Following this are optional parameters for the function. Each parameter is enclosed with square braces, and starts with the parameter name, then an equals, and followed by the data (which can include spaces).

Note that you can insert InScript variables anywhere within a short function tag.

```
%%function [param1=!!address!!]%%
```

Full function tag

A full function tag gives you more features than a short function tag. Their primary advantages are:

- You can enclose large sections of text
- You can insert short-function tags as parameter values

First, let's look at a full function tag:

```
<inscript func="function" param1="some data" param2="other data">
Lots of body data
</inscript>
```

Note that if you do not want any body data you can shrink this to:

```
<inscript func="function" param1="some data" param2="other data"/>
```

The full function tag looks like any XHTML tag. It starts with the tag name of 'inscript', and must then be followed by a function (`func="name"`). Optional parameters can then follow this, in standard XHTML style.

You can insert short function and variable tags anywhere in the full function tag:

```
<inscript func="!!myfunction!!" param1="%%post_date%%"/>
```

Special parameter values

Additionally, there are two special parameter values:

- `%1` – Contents of full tag body
- `%2` – Contents of WordPress element (e.g. the body of your post, the body of the date etc)

For example, if you want to use the data inside an InScript function as a parameter to the function, you would use the %1 value:

```
<inscript func="format_highlight" code="%1">
<?php echo "Hello world"; ?>
</inscript>
```

This would set the value of 'code' to '`<?php echo "Hello world";?>`'.

The second special value refers to the whole of the WordPress element where the tag is being used. For example, if the tag is inside the content of a post, then `%2` refers to the whole of the post content. The reason for this becomes apparent when tags are used as custom fields. In this instance, `%2` refers to the

whole of item with a custom field. For example, if you assign a field to ‘the_title’, then %2 will contain the contents of the current post title.

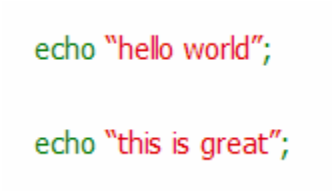
Special encoding parameters

In addition to the special parameter values, there are some special encoding parameters that affect the way the content of the tag is processed.

Before these are detailed, an example is required to highlight a problem. The following InScript tag:

```
<inscript func="format_highlight" code="%1">
echo "hello world";
echo "this is great";
</inscript>
```

Would produce this output:

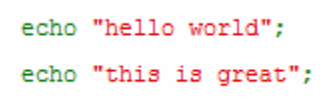


At first glance this may seem correct. However, each line has been turned into a paragraph and, even worse, the quotes have been converted into fancy quotes – if someone copies this code then it will not run. Less importantly, the font is a proportional font, which is not good for code.

These problems are the result of the inbuilt formatting features of WordPress, specifically wptexturize and wpautop. We can disable these features globally from the InScript options page, but this will affect everything. Instead, it would be better to disable them for the section of code itself. Normally this is not possible, but with a special InScript parameter we can achieve it.

```
<inscript func="format_highlight" code="%1" wp="off">
echo "hello world";
echo "this is great";
</inscript>
```

Note the argument wp=”off”. Now when we look at the result:



The code is exactly as we would expect.

Any InScript tag, whether short or full, can have the argument ‘wp’. This can be set to:

- off – disable paragraph formatting, and texturizing
- p – disable paragraph formatting only, but keep texturize enabled
- texturize – disable texturize only, but keep paragraph formatting
- full – enable paragraph formatting and texturizing

There is also another special InScript argument. If we tried this tag:

```
%%echo [text=<em>This is HTML</em>]%%
```

The result is to echo 'This is HTML and we get

This is HTML

You can see that the HTML within the tag has been picked up by the browser and displayed as HTML. This may be exactly what you want. However, sometimes you want to display it as plain text. To do this you can set an encoding parameter that encodes all HTML entities so they will not get interpreted by the browser.

```
%%echo [text=<em>This is HTML</em>] [ent=on]%%
```

Now when we look at the output:

This is HTML

The HTML entities, such as the angle brackets, have been encoded so that they are displayed as angle brackets and not interpreted as HTML.

WHERE TO PUT TAGS

We've looked at what InScript tags are, now let's see where we can put them. Tags can be used in several places:

- Embedded directly into text
- As a post-specific custom field
- As an InScript global custom field

Embedding tags into text has already been discussed and will not be covered further here. Before we look at the custom fields let's take some time to talk about WordPress actions and filters.

The WordPress plugin framework exposes many 'hooks'. These are ways for a plugin to attach to a particular piece of data or operation, and manipulate it. Almost everything that appears on a WordPress page has been passed through many plugins. The most popular of these plugins is the inbuilt texturizer, which transforms basic text into fully-formatted paragraphs.

These hooks are split into two categories:

- Actions
- Filters

An action is 'fired' when a specific event occurs. For example, when you edit a post and press the 'save' button, an action is fired. A plugin can register an interest in this action, and then manipulate the post.

A filter is slightly different in that it is not 'fired' on occurrence of an event, but simply manipulates data. For example, when WordPress wants to display the content of a post, it passes the content through a filter. A plugin can register an interest in this filter and so manipulate the post.

Now, the reason for this explanation is that it affects where InScript runs. By default InScript will process tags in the content of a post, and this is because it has registered an interest in the appropriate filter. You can customise InScript to register interests in any of the WordPress actions and filters.

More specifically, if you wish to manipulate the whole of a WordPress element (such as the content, or a post title, or a date), then you must tell InScript what it is that you want to modify.

Post-specific custom field tags

Post-specific tags are assigned to a post by adding custom fields. These custom fields must be in a special format: the word '`inscript_`' is appended with the name of the filter or action you want the tag to affect.

For example, to add an InScript field tag that affects the post content, you need to add a field as follows:

```
inscript_the_content
```

If you want to affect the post title:

```
inscript_the_title
```

You can have as many fields as you wish, and the same filter or action can appear several times. In this instance, the tags are executed in the order in which the fields occur.

Note that you must have hooked the appropriate filter or action in the InScript options page. This will be explained later.

Here is an example custom field that changes the time of a post:

Notice how the custom field name is 'inscript_the_time'. This tells InScript that the data should be applied to the filter 'the_time'. The data itself is an InScript tag that runs the function 'time_since'. This function modifies a time so that it display how long ago the time is from now. This function takes a parameter 'time'. In the example it has been set to %2 – this is where the special parameter values come into play.

Remember previously that %2 refers to the whole of the element being modified? In this case, the element being modified is 'the_time' (the time of a post). We use %2 to tell InScript which parameter the body of 'the_time' (the time value of a post, for example '6th March 2005') should be inserted into. In the above example, this value is inserted into the 'time' variable.

A custom field can contain anything, not just InScript tags. For example, if you want to change the title of a post you could do this:

```
inscript_the_title = My title is <em>'%%post_title%%'</em>
```

Global custom fields

Custom fields are managed by WordPress, and are assigned to a post by it's author. They are used by InScript to allow functions to be performed against a specific post. However, sometimes it would be nice to perform functions everywhere, regardless of the current post. We can do this using global custom fields.

Global custom fields are managed by InScript from the InScript options page, detailed later. Here you can create, delete, and update the global custom fields. A global custom field is exactly the same a post-specific custom field, but it is just applied globally.

For example, to change the time of every post create this global custom field:

```
inscript_the_time = %%time_since [time=%2]%%
```

Now every post on your blog will display how old the post is, rather than when it was created.

Disabling InScript for a post

There are times when you may want to disable InScript on a single post. You can do this by adding the following custom field to a post:

```
inscript_disable = true
```

Disabling InScript for sections of text

Sometimes disabling InScript for a post is not enough, and you may find that you get unwanted side effects. For example, you might have a custom field that changes the date of a post. This will effect not only the date in the post itself, but also the date of that post everywhere else – the post meta-data, the archives, search results etc. This is generally fine, but in some circumstances, you want the original data to be displayed.

In these situations, there is no alternative but to make a modification to the theme template files. Unfortunately, it is not possible for InScript to determine where it is being called from. That is, it knows it is modifying the date of a post, but it doesn't know if it's the one you want changed.

You can disable InScript from within your theme by adding a call to:

```
<?php inscript_disable (); ?>
```

Conversely, you can then re-enable InScript with:

```
<?php inscript_enable (); ?>
```

Using these two functions, you can wrap sections of your theme and prevent InScript from affecting the data.

CONFIGURATION

Configuration of InScript takes place from the InScript section of the WordPress 'options' administration page.

The InScript options page is split into several sections:

- InScript
  - General
  - Variables
  - User restrictions
- Hooks
  - Filters
  - Actions
  - Global custom fields

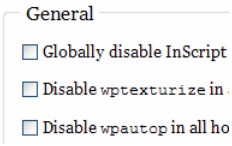
Each of these will be looked at in turn. The default configuration of InScript is likely to suffice for most users.

GENERAL OPTIONS

This section gives you the ability to:

- Disable InScript
- Disable `wptexturize`
- Disable `wpautop`

Disabling InScript does not disable the plugin itself, but disables the processing of tags, which are quietly stripped away. You would use this if you wanted to stop temporarily InScript but not have the tags appear everywhere.



The other options allow you to disable the WordPress `wptexturize` and `wpautop` functions. These functions process text and convert it into a more attractive format. The `wptexturize` function performs fancy transformation of certain characters, such as replacing straight quotes with curly quotes. The `wpautop` function inserts proper paragraphs.

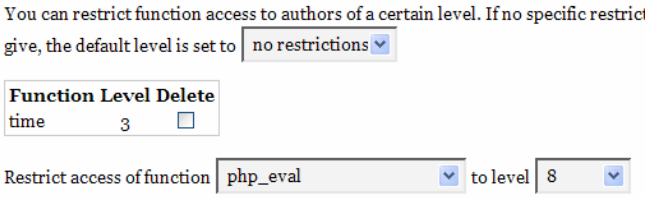
Why would you want to disable these? Well, although very useful functions they do sometimes cause unwanted side effects. You can disable them here, and then use your own formatting, as desired.

USER-LEVEL RESTRICTIONS

Sometimes it may not be desirable to provide access to certain InScript functions. For example, the '`php_eval`' function allows you to execute any PHP code. This can be a potential security risk if you do not trust every author.

InScript allows you to assign restrictions to every function. These restrictions are based upon the standard WordPress user levels.

To add a restriction you select the function and choose a level:

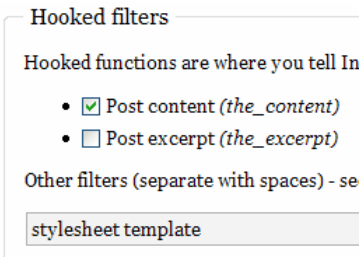


Here the function '`php_eval`' has been restricted to users who have a level of 8 or higher.

You can also assign a default level that is applied to every function that is not already restricted.

HOOKS

Configuring hooks is a simple process. By default, the InScript plugin will attach to the content filter, and you may not even need to configure it further. If you look at the InScript options page, you will find a section for hooks:

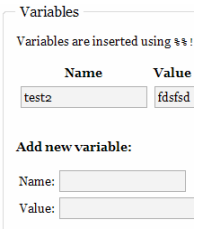


The checkboxes represent the standard filters and actions that you would normally want to use. The text box is for advanced usage, and allows you to specify any of the other WordPress filters and actions (a link to the full list in the WordPress Codex is provided from the options page).

Here you can see that the post content has been hooked, and so InScript tags can be used to manipulate post text. Also, two custom filters have been added as an example – here they are 'stylesheet' and 'template'.

VARIABLES

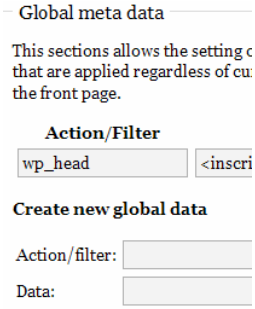
Variables are pieces of data that are assigned a name.



Variables can contain any text, HTML, or even other InScript tags and variables.

GLOBAL CUSTOM FIELDS

Global custom fields are added in a similar way to normal custom fields. However, in this instance you don't need to prefix the name with `inscript_`. The value still follows the same rules as custom fields, and you can insert any text, HTML, or InScript tags.



Note that any filter or action you use in a global custom field must also be hooked in the appropriate section of the options page.

THINGS YOU MIGHT WANT TO HOOK

Here is a list of common filters you might want to hook

- `the_time` – To modify appearance of a post's time
- `the_title` – To modify appearance of a post's title
- `the_content` – Modify the contents of a post
- `the_excerpt` – Modify the excerpt of a post
- `option_blogdescription` – The blog description
- `option_blogname` – The blog name

Here is a list of some actions you might want to hook

- `wp_head` – Insert data into the 'head' section
- `wp_footer` – Insert data into the 'foot' section
- `wp_meta` – Insert data into the meta-section of the sidebar

FUNCTION LIST

Finally we get to the list of functions. Many of these functions are straight copies of standard PHP or WordPress functions, but are repackaged for InScript. In these cases, you can find further documentation on the PHP or WordPress websites.

AUTHOR FUNCTION

Most author functions take two optional parameters: id or login. This is the ID of an author, or the login name of an author. If no parameter is given then the author of the current post is used.

author

Params: `idmode` (optional)
Returns the current author's name using the same parameters as `the_author()` function in the Codex.

author_login, author_firstname, author_lastname, author_nickname

Returns the appropriate name information about the author.

author_description, author_id, author_email, author_url

Returns the author description, id, email, or url.

author_icq, author_aim, author_yim, author_msn

Returns the appropriate instant messenger information about the author

author_fullname

Returns the authors name in the format configured in the author's profile.

author_numposts

Returns the number of posts made by the author (current, ID, login)

Examples

You could create a WordPress page containing statistics of all your blog authors:

```
<h3>Blog statistics :</h3>

%%author_fullname [id=1]%% has created %%author_numposts [id=1]%% posts
%%author_fullname [id=2]%% has created %%author_numposts [id=2]%% posts
```

BLOG FUNCTIONS

Blog functions return information about your blog. This information is the same as returned from the `bloginfo` function.

blog_name

Returns the blog name (i.e. the title of your blog)

blog_description

Returns the blog description as configured

blog_url

Returns the proper URL to your blog

blog_admin

Returns the administrators email address

blog_version

Returns the version of WordPress currently in use

blog_template_url / blog_template_directory

URL to current template directory

blog_stylesheet_url / blog_stylesheet_directory

URL to current stylesheet directory

blog_wpurl

Returns result of `bloginfo ('wpurl')`

Example

You could make a direct link to the administrators email address as follows:

```
You can contact the <a href="email:%%blog_admin%%">admin</a>
```

COMMENT FUNCTIONS

These functions return comment information.

comments_list

Params: `before`, `after`, `format`, `count`, `limit`, `type` (all optional)
Returns a list of comments. By default the list will be an unordered list, with each comment being a link to the post. You can modify this as follows. The '`before`' and '`after`' parameters can be given text to display before and after the comments (default '`<ul>`' and '`</ul>`' respectively). The `format` parameter is what gets displayed for each comment – the special words `$authorurl`, `$post`, `$title`, `$author`, `$comment`, will be replaced with the appropriate information. The '`count`' parameter defines how many comments (5 default), and '`limit`' defines how much of the comment text to return. You can choose random comments by setting '`type`' to '`random`'.

comments_total

Params: `post`
Returns the total comments for a given post, or across the whole of the blog

FILE FUNCTIONS

These functions return file information.

file_size

Params: `file` (required)
Returns the size of the file, in bytes.

file_modified

Params: `file` (required), `format` (optional)
Returns the date the file was last modified, in the format configured in WordPress. You can change this format with the 'format' parameter.

file_created

Params: `file` (required), `format` (optional)
Same as file_modified, but returns the date the file was created.

FORMATTING FUNCTIONS

The formatting functions allow you to modify the format of some text.

format_highlight

Params: `code` (optional), `file` (optional)
Highlights code using PHP's in-built highlighting function. If you are running on PHP4, the code will use `<font>`, while PHP5 will use `<div>`. The code to highlight can be given as the parameter '`code'`, or you can refer to a file with the parameter '`file'`. Note that the file will be relative to the root directory of your WordPress installation.

format_markdown

Params: `text` (required)
Formats the text using the Markdown syntax. If you are using this function you should disable the WordPress formatting functions.

format_textile

Formats the text using the Textile syntax. If you are using this function you should disable the WordPress formatting functions.

format_geshi

Params: `code`, `file`, `lang`, `line`, `class`, `head`, `foot` (all optional)
Highlights code using the Geshi highlighter. This is a very powerful highlighting tool, and can highlight over 50 languages, including HTML, CSS, and PHP. The code to highlight is passed either as the parameter '`code'`, or as a file relative to the WordPress installation '`file'`. You tell Geshi what language the code should be highlighted as using the parameter '`lang'`. By default this will be '`html'`.

You can enable line numbers with the 'line' parameter. Set to '`0`' it will add simple line numbers, set to any other number will add line numbers, and will include alternate colour highlighting every line number you specify. If you add the '`class'` parameter, then the code will be given the specified CSS class. If you do not add this, the code will use default CSS styles. You can also add '`head'` and '`foot'` that will be displayed at the head and foot of the code. This could be used to add a title and link to the code.

Example

You can highlight different sections of text as follows:

```
<inscript func=”textile” text=”%1”>
This will be formatted using Textile
* list 1
* list 2
</inscript>
<inscript func=”markdown” text=”%1” wp=”off”>
But this is Markdown
* list 1
* list 2
</inscript>
```

Note that %1 is a special value, indicating that the 'text' parameter should be passed the entire contents of the body of the function (everything between <inscript> and </inscript>)

HEAD FUNCTIONS

The head functions allow you to insert HTML `<head>` tags. These can add extra stylesheets, as well as providing meta-information for search engines.

head_title

Params: `title` (required)
Inserts an HTML title with value '`title`'

head_style

Params: `url` (required)
Adds a link to the CSS stylesheet specified by '`url`'

head_import

Params: `url` (required), `media` (optional)
Adds a link to the CSS stylesheet specified by '`url`', but uses the '@import' method. You can specify an optional media-type for the stylesheet. The default is '`all`'.

head_keywords

Params: `keywords` (required), `alt` (optional)
Add a keywords item using the text in '`keywords`'. If this is empty it will use the text in '`alt`'.

head_description

Params: `desc` (required), `alt` (optional)
Adds a description item using the text in '`desc`'. If this is empty it will use the text in '`alt`'.

head_meta

Params: name (required), content (required), equiv (optional)
Add a HTML meta item, using the name '`name`', the content '`content`'. You can add an optional '`equiv`'.

Example

If you want to automatically set the keywords and description for your blog then create two global custom fields:

```
wp_head=<inscript func="head_keywords" keywords="%%post_categories%%" alt="wordpress blog, great"/>
wp_head=<inscript func="head_description" desc="%%post_excerpt%%" alt="My super wordpress blog"/>
```

Note how the '`alt`' parameter has been specified. When a single page post is being viewed then the keywords and description will be set according to the post. However, on a multiple-post page (such as the front page), it doesn't make sense to do this, so the keywords and description are set to the alternative value.

HTTP FUNCTIONS

These functions provide access to some HTTP variables, such as the name of browser in use.

http_host

Returns the hostname of the user requesting the page.

http_ip

Returns the IP address of the user requesting the page.

http_agent

Returns the browser name of the user requesting the page.

http_referer

Returns the URL which the requesting user may have used to browse to the page

PHP FUNCTIONS

These functions provide access to certain PHP functions that allow you to include other PHP code.

php_version

Returns the version of the PHP software currently being used.

php_info

Params: `what` (optional)
Returns information from the `phpinfo` command. This allows you to get full system information, as well as configuration settings. The optional argument takes the same values as the PHP function (general, credits, configuration, modules, variables, license) and defaults to displaying everything.

php_eval

Params: `code` (required)
Executes the '`code`' as PHP. This is a very powerful function, and should not be accessible to people you do not trust (i.e. you should not let people embed this in comments). The code can be any fragment of PHP. The function will automatically add a trailing semicolon, and is smart enough to unmangle '`< ?`' into '`<?`'

php_include

Params: `file` (required)
Similar to `php_eval`, but this allows you to include the contents of any PHP or HTML file. If the `file` includes PHP code then it will be executed. Again this is very powerful and can lead to all sorts of fun, and all sorts of damage. You can do things such as embedding forums within a post. The file must be local to your website.

php_virtual

Params: `file` (required)
Very similar to `php_include`, but this one allows you to embed SSI and cgi-bin files into your blog.

Example

You may have a custom shopfront solution that you want to include within one of your blog posts:

```
You can buy this item at my shopfront <inscript func="php_include" file="/shop/displayshop.php"/>
```

Alternatively you may just want to execute a function:

```
The current year is %%php_eval [code=echo date ('Y')]%%
```

POST FUNCTIONS

These functions provide access to post information. Unless specified otherwise, post functions can be given these parameters:

- `url` – refer to a post using the relative URL (i.e. /2005/05/03/mypost)
- `post` – refer to a post using its ID, or '`all`'. If you given '`all`', the function are applied to every post on the current page, with the results joined and separated by the '`join`' value (defaults to a space).

When neither parameter is given, the post will default to the current post (or nothing if no current post).

post_page

Returns the page number of the current post (no parameters accepted).

post_custom

Params: `url` & `post`, `key` (required), `merge` optional
Returns the custom-fields with name '`key`' that are associated with the post. The values will be joined together using the value of '`merge`', or a space if no value is given.

post_id

Params: `url` & `post`
Returns the post ID of the given post

post_date

Params: `url` & `post`, `format` (optional).
The date format will be the default system format unless the '`format`' parameter is given, which is in PHP date format.

post_content

Params: `url` & `post`
Returns the content of the post, without any plugin manipulation (i.e. it will not be formatted)

post_title

Params: `url` & `post`
Returns the title of the post

post_excerpt

Params: `url` & `post`
Returns the post excerpt

post_modified

Params: `url` & `post`, `format` (optional)
Returns the date the post was last modified (see '`post_date`' for format details)

post_guid

Params: `url` & `post`
Returns the WordPress GUID for the post. This is a unique URL referring to the post

post_total

Returns the total number of posts

Post_categories

Params: `url` & `post`, `format` (optional), `merge` (optional)
Returns the categories for the post. By default, this will include the category names, separated with a comma. You can change the separation string by setting the parameter '`merge`'. You can change the text by changing the parameter '`format`'. The format parameter can include any text, with the special words `$name`, `$id`, `$desc`, and `$nice` being replaced with the category name, ID, description, and nice-name respectively.

The following functions use the Coffee2Code 'customizable-post-listings' plugin. They take the same parameters and full documentation is found at the Coffee2Code website. Note that you must have the plugin installed and enabled for these functions to work.

post_c2c_recent

Returns recent posts. Parameters as defined on c2c website

post_c2c_random

Returns random posts. Parameters as defined on c2c website

post_c2c_recently_commented

Returns recently commented posts. Parameters as defined on c2c website.

post_c2c_recently_modified

Returns recently modified posts. Parameters as defined on c2c website.

Examples

If you want to display how long ago a post was modified:

```
<inscript func="time_since" time="%%post_modified%%"/>
```

RANDOM FUNCTIONS

These functions provide random data.

random_file

Params: `dir` (required)
Returns a filename chosen from random from the given directory. Note that the filename will not include any path information, and the directory searched will be relative to the WordPress installation.

random_word

Params: `file` (optional), `text` (optional)
Randomly choose a word for a section of '`text`', or from the '`file`'.

random_line

Params: `file` (optional), `text` (optional)
Randomly choose a line from a section of '`text`', or from a '`file`'. Note that a line is considered to be any section of text that ends with a newline, or a carriage return and newline.

Example

Randomly insert a word from a variable:

Variable:

```
word_list = cool super great ok fine excellent
```

Tag:

```
This post is %%random_word [text=!!word_list!!]%%
```

Will produces variations of 'This post is cool', 'This post is great', etc.

STRING FUNCTIONS

These functions allow you to modify words.

echo

Params: `text` (required)
Simply echos (displays) the contents of '`text`'.

str_toupper

Params: `text` (required)
Converts all characters in '`text`' to uppercase.

str_tolower

Params: `text` (required)
Converts all characters in '`text`' to lowercase.

str_firstupper

Params: `text` (required)
Converts the first character of the first word to uppercase.

str_allupper

Params: `text` (required)
Converts the first character of all words to uppercase.

str_isbn

Params: `isbn` (required)
Returns a URL to a book at Amazon, given it's ISBN number.

str_rot13

Params: `text` (required)
Rotates each character 13 positions along the alphabet. This is a standard computer method for obscuring words.

str_shuffle

Params: `text` (required)
Shuffles the characters in the '`text`' at random.

str_wordwrap

Params: `text` (required), `len` (optional), `break` (optional), `force` (optional)
Wraps the given text to the given length (or 75 characters by default. The value of '`break`' is used to break the words, or a newline is used by default. If '`force`' contains any value then the function will force long words to be broken.

str_striptags

Params: `text` (required)

Removes all HTML tags from the `text`.

str_reverse

Params: `text` (required)
Reverse the `text`!

str_md5

Params: `text` (optional), `file` (optional)
Produces an md5 checksum of the given `text` or `file`.

str_obscure_email

Params: `email` (required)
Obscures an email address so that spambots will not detect it. The email will be transformed as follows:

```
test.name@email.com => test dot name at email dot com
```

str_leet

Params: `text` (required)
Oh dear. This does a simple translation of the text into Internet '`leet`' speech. If you don't already know then you are better off not finding out!

str_wordcount

Params: `text` (required)
Returns the number of words contained in the `text`.

str_repeat

Params: `text` (required), `count` (optional)
Repeats the given `text` the specified number of times (or once if no `count` is given).

Str_bytes

Params: `bytes` (required)
Formats a numeric string into the largest units of bytes. For example, 1024 bytes would appear as '1 KB'.

Examples

For example, if you wanted to ensure that some text was only 70 characters wide (useful for `<pre>` blocks):

```
<pre>
<inscript func="wordwrap" text="%%post_excerpt%%" len="70"/>
</pre>
```

If you wanted to add a word count to a post, add this as a custom field:

```
inscript_the_content=Word count=%%str_wordcount [text=%2]%%<br/>%%echo [text=%2]%%
```

SYSTEM FUNCTIONS

These functions provide access to the server system on which your WordPress blog is running.

func_system

Returns full system information (OS, version, release etc)

func_system_os

Returns OS of the server

func_system_hostname

Returns the server hostname

func_system_release, func_system_version

Returns the release or version of the server's OS

func_system_machine

Returns the type of machine the server is running on

func_system_mysql

Returns MySQL information

func_system_server

Returns information about the webserver

func_system_ip

Returns the server's IP address

func_system_port

Returns the port the webserver is running on

func_system_root

Returns the root directory of the webserver

Func_system_date

Params: `format` (optional)
Returns the server date in the given PHP date `format`, or the system default if none given.

THEME FUNCTIONS

These provide access to the current theme.

theme_name

Returns the current theme name

theme_description

Returns the current theme description

theme_author

Returns the author of the current theme

theme_version

Returns the version of the current theme

TIME FUNCTIONS

These provide functions to handle dates and times.

time_since

Params: `time` (required)
Returns the time since the given `time`. This will be returned as two values: years and months, months and days, days and hours, hours and minutes. If the given time is in the future then it will be the difference between now and the future time.

time_fuzzy

Params: `time` (required)
Adaption of the 'Time of day' plugin. Given a `time` it will return a fuzzy word describing it. For example, if the time is 3am, the word used will be 'terribly early in the morning'.

Examples

To change the date format of a post add this custom field:

```
inscript_the_time=<inscript func="time_since" time="%%post_time%%"/>
```

WordPress functions

These provide access to certain WordPress functions. The parameters are all the same as can be found in the WordPress Codex.

wp_getcalendar

Returns a calendar of blog activity

wp_register

Returns a registration login link

wp_loginout

Returns a login/logout link

wp_get_archives

Returns archives

wp_link_pages

Returns link pages

wp_list_pages

Return pages

wp_get_recent_posts

Returns recent posts

wp_get_links

Returns links

wp_list_authors

Returns authors

wp_list_cats

Returns categories

OTHER FUNCTIONS

jumpto

Params: `list` (required)
Returns a drop-down listbox of links that refer to pages within a multi-page post. When a link is selected, the current page will be changed accordingly. The parameter '`list`' should contain the links, with each link on a separate line. Each line should be in the format '`page=name`'. For example:

```
1=Title of page 1
2=Title of page2
```

list

Params: `list` (required)
This will display an unordered list, with a list item for each line of '`list`'.

linklist

Params: `list` (required)
Link the previous function, but '`list`' is of the form '`name=url`'. The list will be presented as a list of URL links. For example

```
Some link=http://www.someplace.com
Another link=http://www.anotherplace.com
```

google_adsense

Params: `code` (required)
Allows the insertion of Google AdSense code. The code will be automatically escaped and will not be affected by WordPress formatting (there is no need to switch off formatting). Additionally, the function will ensure the adherence to the Google terms & conditions by only allowing three ads per page. Any further calls to this function will be silently ignored.
The value of 'code' should be the full code as returned by Google.

EXAMPLES

Here are some more examples of using InScript

INCLUDE RANDOM IMAGE

You can insert a random image into a post by adding this:

```
<img src=”/wp-contents/images/%%random_file [dir=/wp-contents/images/]%%”/>
```

Here the tag has been inserted inside a normal HTML tag – you can place InScript tags anywhere, either as whole elements on their own, or as parts of other elements.

The `random_file` function randomly selects a file from a directory. In this instance, we tell it to look in a directory of images. It will pick one and return the filename. The resulting HTML will look like this:

```
 <img src=”/wp-contents/images/someimage.jpg”/>
```

Every time anyone looks at the post, the image will be different.

AUTOMATIC MD5 CHECKSUM

A popular way to release software on the internet is to provide an MD5 checksum for the file. This checksum allows a user to verify that the file they have downloaded is the file that the author released.

Normally this checksum is generated with an external program, and must be updated every time the software is changed. Instead, we can let InScript automatically generate it for us:

```
File checksum: %%md5 [file=mysoftware.zip]%%
```

REFERRING TO ANOTHER POST

You may want to include the excerpt from another post inside a post. We can get InScript to do this for us:

```
%%post_excerpt [url=/2005/05/04/mypost]%%
```

SPEAKING IN ESREVER

An important feature here. Using some of the more esoteric functions, you can do something like this:

```
<inscript func=”reverse” text=”%1”>
This is an example sentence.
</inscript>
```

And the result will be:

```
.ecnetnes elpmaxe na si sihT
```

Oh, the hours of fun you can have.

FORTUNE COOKIE

A simple use of InScript:

```
%%random_line [file=cookies.txt]%%
```

This will randomly select a line from a file and display it. If this file happened to contain fortune cookie quotes, then you have an instant cookie generator.

AUTOMATIC WORD PUZZLES

Yet more fun.

```
<inscript func=”shuffle” text=”%%random_line [file=words.txt]%%”/>
```

This will pick a random line from a text file and then shuffle the letters to form a word puzzle.