User Guide

How To Get The Most Out Of The PoliMorf WordPress Theme

v1.0
Bournemouth, UK, June 2018
M. H. Corbett

Contents

Introduction 6
Getting Started With PoliMorf 6
Installing PoliMorf in WordPress *6*
What is so special about PoliMorf? 8
How is this different from using Shortcodes? *8*
The Author 9
The Basics 11
Setting up a sidebar, header menu and basic footer 11
Manual Styling 13
Default Schematics 14
Default Header 14
Adding more links to a custom header menu *15*
Adding a logo custom header menu *15*
Default Post Content 17
Default Page Content 18
Default Footer Content 18
Other format fields 19
Adding & Changing Content 21
Extra Content 22
Meta Data 25
Inserting Content into WordPress default content 26
Adding Boxes 30
Examples 32
More Elements To Play With 34
PoliMorf Substructure 34

Creating Sections 35
Adding Quotes 37
Adding Menus 38
Post Lists 39
Page and Post Custom Meta 39
Alternative Featured Image url *40*
Custom Excerpt *40*
Additions 40
Font references *41*
Scroll Reveal effects *41*
Stylesheet references *42*
Background image link references *42*
Header and Footer Content *43*
Custom Classes, Fonts, Effects and Styles 44
Preset Custom Classes 45
Remote Data Calls 46
Remote calls at Global and Schematic Level 47
Configuring Remote Assets 48
User Policy *48*
Bucket Policy *49*
PoliMorf Settings->Cloud 50
Remote Syntax 50
Top Level Remote Call *51*
Schematic command level call *51*
Remote link and image calls *51*
Alternative Featured Image Url remote call *53*
Custom Excerpt remote call *53*
Remote BACKGROUND call *54*
Inline Styling and Remote Stylesheet Reference *54*

Header and Footer Content Script Reference 55
Custom Functions and Attributes 56
PoliMorf Functions 56
Linking custom functions to schematic command processing 57
Useful PoliMorf functions 58
pcom_get_strings_syntax_separator 58
pcom_process_command_open_close_syntax 58
Custom Attributes 59
Custom syntax 63
Appendix A: Command List & Constants 66
Identifiers 66
Command List 66
Schematic Tags 67
NAV 67
MENU 69
TEXT 70
CONTENT_DEFAULT 73
CONTENT_META 73
CONTENT_INSERT 74
QUOTE 75
SECTION 77
BACKGROUND_IMAGE 77
COPYRIGHT 78
N-BOX (2, 3 and 4) 79
STYLE 79
STYLESHEET 80
Number of Widgets 80
REMOTE_SCHEMATIC_AWS 80
REMOTE_IMAGE_AWS 81

EFFECTS *82*
FONT *82*
HEADER_CONTENT *83*
FOOTER_CONTENT *84*

.

Introduction

PoliMorf is a WordPress-based Theme designed to allow a wide range of on-screen configurations for a website using the WordPress.org content management system.

PoliMorf also has the power to let you call these configurations from the Cloud rather than store them on your hosting database. This allows you to decide what data you want to store locally or remotely, and whether you choose to edit your website data and layout in the Cloud or at source.

To do this, the theme uses a schematic language that simplifies design into a linear layout. This user guide was written to help you get familiar with that schematic language and to show you how to extend it or change it.

Getting Started With PoliMorf

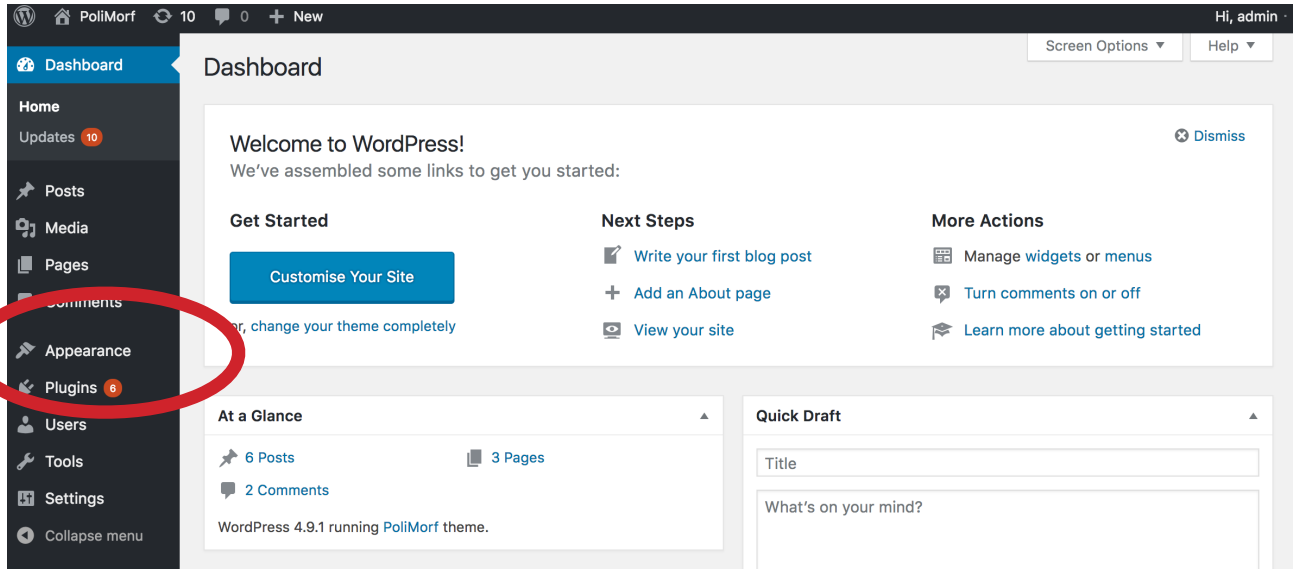
If you have downloaded the theme you will be able to see all the different folders and files where various functions and classes are called. Don't worry. You will not need to edit any of these files if all you are interested in is designing layouts. If, however you are curious you can have a look.

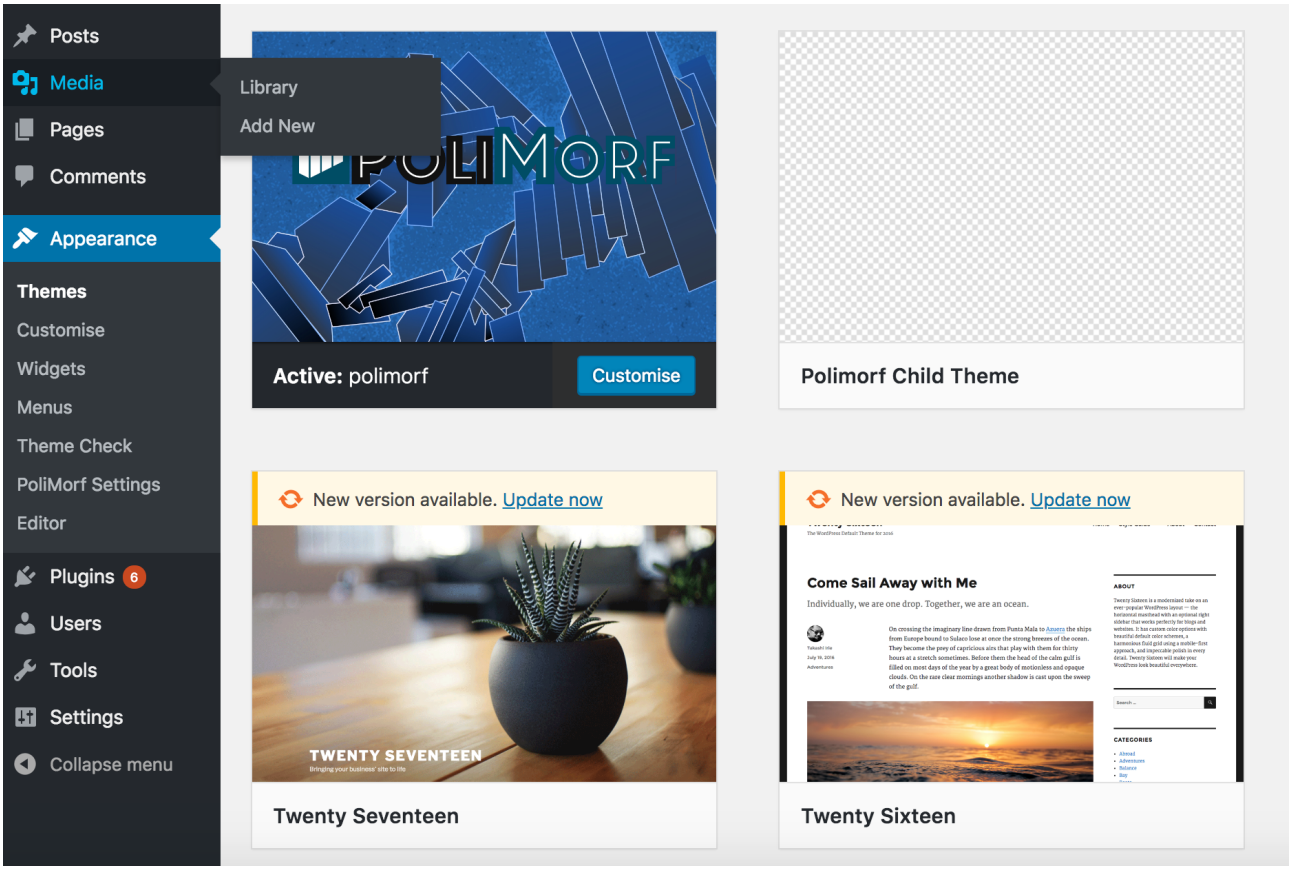
The first thing to do if you want to run PoliMorf on your WordPress site is to install the theme.

Installing PoliMorf in WordPress

First off, to install a theme on WordPress the best way to do it is from the Themes section in the WordPress administration area, or the WordPress Admin for short.

If you go there now and are in the main Dashboard area you should see a menu on the left. The tab that says "Appearance". If you click on this and select "Themes" you should go to a page where there will be a number of listed themes. If you have just installed WordPress there will be a number of "Twenty…" type themes.





To activate one of these themes you would hover over the theme image and follow the steps.

To install you need to click the "Add New" button at the top and then "Upload Theme".

.

WordPress prefers that you load a **.zip** file. If you have downloaded WordPress and your computer has automatically extracted PoliMorf as a folder you will need to re-zip it. Or if you have the original .zip intact you can use this.

WordPress will upload and ask you to customise the theme using the Customizer (default language is US English). You can do so if you wish but make sure you read the rest of this manual or you might get confused as to what is going on.

If you don't wish to customise just yet, accept and install.

You are then ready to go!

If you have already installed PoliMorf, have changed it and want to reinstall from fresh, select any other theme and activate it. Then click on PoliMorf and delete it, using the link that appears. Then reinstall and activate. And don't worry, you won't lose any setting. If you have added custom functions (which you can read about in later sections), save this file somewhere so you can copy the information back into your new install.

What is so special about PoliMorf?

Rather than answer this as a rhetorical question, the most immediate answer is that PoliMorf **does not have specific template pages** beyond those that are required for a WordPress theme.

If you have used other themes you may find that there is a page template for a signup page, or a gallery, or an introduction page.

What these templates do is allow you to add lots of information but stay within the structure of the template. So if there is a sidebar on the page you may not be able to switch it off easily.

With PoliMorf, your default templates are defined as a list and they are easily changed. The idea is that you design using a "meta-template list". You scope out what you would want and list it, then the theme interprets that and produces various html elements.

How is this different from using Shortcodes?

Shortcodes in WordPress are content entries in the form of commands within square brackets e.g.:

"[shortcode arg1=".." ..] [/shortcode]

Shortcodes are used to add special elements within WordPress content areas, such as the main content or sidebar. A shortcode may add a flashing image link, or a signup form, or even a slideshow.

A lot of WordPress themes and plugins use shortcodes, and PoliMorf allows them to be used as well. That said PoliMorf does not rely on shortcodes. The schematic language and syntax can be modified to be nothing like a shortcode and it will still work. So the schematic language is more broadly defined.

If you only ever use PoliMorf to define a basic structure and use shortcodes for all other things, that's fine. PoliMorf supports that. It even supports extra modifications through Custom Functions, but that's for later.

The Author

PoliMorf was created by Micky H Corbett (above), a scientist/engineer/ entrepreneur, who has wide-ranging experience in all manner of things including:

- Building and testing spaceship engines that have gone on to fly in space.
- Testing flight control systems for planes and helicopters.
- Creating structures at a nanoscale to investigate better computer memories.
- Programming in multiple languages (it comes with the territory) and always trying to improve and learn!

PoliMorf and PoliMorfic are ventures of the space/aerospace company Corvos Astro Engineering Ltd who are based in Bournemouth, U.K. Developing software is common place in engineering and since the company website needed to be maintained it was a natural shift to programming a WordPress theme. I had already been doing this for a number of years.

The idea for PoliMorf came through learning about the improvements in Cloud computing access, and seeing that a WordPress theme could be made to act more decentralised because of it.

There are lots of ways to achieve this: PoliMorf took the route to eliminate the need for custom templates so that each page can be resemble a static page. It means more

work for the user but also more control. You can make each page look however you want but without needing to write full HTML markup.

And rather than go fully serverless with the infrastructure, PoliMorf is a balance between WordPress and AWS functionality. You get to keep the handy functions of WordPress and get the security of AWS.

.

1
The Basics
Just Another WordPress Site

You have uploaded PoliMorf to your site and are faced with a bland looking full-width layout with a minimal menu and footer. So what now?

Setting up a sidebar, header menu and basic footer

The first thing to do if you want to show a sidebar on the right-hand side of the page is to login to your admin page (the backend) and go to the Sidebar Format option in *PoliMorf Settings -> Styling*. Type the following:

```
%%WIDGETS::
```

This is an example of a schematic command.

There is a command beginning identifier “%%” and a command end identifier “::”. In between, there is a command, in this case “WIDGETS”.

What this command does is as follows:

- It tells the theme that you want a sidebar visible for all pages and posts. It is applied on all pages and posts because you are filling in the global option Sidebar Format. You will have to overwrite individual posts and page formats if you don’t want a sidebar to show. Or you could use css to hide it.
- The command activates the Widget Area so you can use any WordPress widgets along with any plugins you have. These appear in the Widgets section, which comes as standard in the WordPress admin section. There are 5 widget areas by default that can be specified by adding 1 to 5 after the command. If no number is specified it will default to Widget Area 1.

The next thing is to go to the Header Format option and type the following:

%%NAV::
wpmenu={}:

This tells the theme that you want a navigation bar (which is responsive) and that you want to use the internal WordPress menu. The keyword is “wpmenu” and the open and closed brackets (={}:) is an empty attribute. This will default to the menu set in the Menu section of WordPress.

If you want another menu that you have created (in the same Menu section) use:

wpmenu={ name=“Name of Menu” }:

You can use spaces inside the identifiers (={}:) as these will be trimmed. If you want to be able to style the menu you can use a class attribute:

wpmenu={ name=“Name of Menu” class=”my-class” }:

Wordpress will automatically assign a class to the menu list object (the html ul tag) as “menu-*menu-name-as-slug*” e.g. “menu-name-of-menu”.

Using the class attribute,as above, will assign a class to the div that wraps the ul. i.e.

<ul class=“menu-*menu-name-as-slug*”>...

Lastly, go to the **Footer format** option and type the following:

%%WIDGETS::2
%%COPYRIGHT::

Add footer widget elements to Widget Area 2. Widget Area 1 has been used in this example for the Sidebar.

You should also have a copyright statement in the footer after the widgets. If you want to add sentences above this copyright statement you can use the following schematic:

```
%%WIDGETS::2
%%CONTENT::
TEXT=[
[p]Sentences here ... [/p]
[p]More sentences here ... [/p]
]:
%%COPYRIGHT::
```

The subcommand is “TEXT”, which adds text content in the same format as the main post or page content. The subcommand opens and closes using “=[“ and “]:”.

The tags [p] and [/p] are translated as and html tags. The difference is that the theme will strip any html tags so as to try and prevent any html or script injection (or in other words, hacking). In a later section the list of possible text tags that are used will be described. You can also add more if you like.

Manual Styling

Styles and options can also be manipulated in the WordPress Customizer. To add extra css, use the Additional CSS field in the Customizer.

If you want to add styles on a specific page or post, PoliMorf adds the slug as a class. For example, if the post is called “Interesting Things” the slug may be “interesting-things”.

The body class will then be “.interesting-things”.

2
Default Schematics
How the default layout works

If none of the global settings are set in *PoliMorf Settings* -> *Styling*, the theme defaults to presets.

Default Header

The default header schematic is:

```
%%NAV::
mlk={href="SITE_HOME" name="Home" src="IMAGES/
home_icon.png"}:
searchbar={}:
```

This sets a navigation menu with one entry, a home icon that brings the user back to the home page i.e. your domain name. The link is created using a short-cut constant called "SITE_HOME". If you use this in a href, url or src attribute it will replace this text with the url for home i.e. http://your-site

Similarly, IMAGES refers to the images folder of the theme, where various theme images are used.

A search icon will be displayed on the right-hand side of the header. When clicked, it will expand a search input field below it.

The navigation menu will switch to a responsive icon (three horizontal stripes) placed on the left hand side when the screen width is less than 750px wide. The main menu will slide down and up on clicking the responsive icon. The search bar will be fully expanded when responsive and be placed in this case below the menu.

Adding more links to a custom header menu

If you wanted to customise your own menu, you can add text menu links (mlk) as follows:

```
%%NAV::
mlk={href="link1" name="Link Name 1"}:
mlk={href="link2" name="Link Name 2"}:
mlk={href="link3" name="Link Name 3"}:
searchbar={}:
```

Adding an src attribute will replace the text with an image as in the first example in this section.

Adding a logo custom header menu

If you wanted to add a logo, which will appear on the left hand side of the menu, you use the logo keyword with a src attribute. This is analogous to the css command for background i.e.

```
logo={ src="url of logo" }:
```

If you stored your logo in the Uploads section of WordPress and uploaded the logo in October 2017 (you can check this within WordPress) you can write:

```
logo={ src="UPLOADS/2017/10/logo-name.extension" }:
```

Extension being the file extension such as 'jpg' or 'png'. The shortcut "UPLOADS" refers to the upload section present in WordPress themes. It is the place were you would usually upload images when in the Admin section.

Note that when you add a logo the menu element will be aligned to the right e.g. for the site Corvos Astro Engineering.

Whereas if no logo is used the menu aligns to the left.

You can also use the WordPress custom header function in the Customiser to add a logo image. It will work when the logo keyword is included and will overwrite the logo src.

Note this header menu is achieved using the following schematic:

```
%%NAV::
mlk={href="SITE_HOME" name="Home" src="IMAGES/
home_icon.png"}:
mlk={href="SITE_HOME/sample" name="Sample"}:
mlk={href="SITE_HOME/section-test" name="Section Test"}:
mlk={href="SITE_HOME/image-aws" name="Image AWS"}:
searchbar={}:
```

Default Post Content

If you are adding post format to the PoliMorf Settings page, the default post schematic is:

```
%%CONTENT_META::
%%CONTENT_DEFAULT::
%%PAGINATION::
%%COMMENTS::
```

What this schematic represents is the following:

- Show the post title, author, date and category.
- Show the content i.e. standard WordPress content you write in the post editor.
- Show the next and previous post links as well as linked page pagination i.e. if you have more than one part to a page or post
- Show the comments box.

If for example you removed the "%%COMMENTS::" section, comments or the comment entry form would not be shown, irrespective of internal WordPress comment settings.

If "%%CONTENT_META::" is not present, no title, author, date or category will be shown.

If you are adding an in-page schematic in the page/post custom meta field, the default schematic is:

```
///HEADER:
%%DEFAULT::
///MAIN:
%%CONTENT_META::
%%CONTENT_DEFAULT::
%%PAGINATION::
%%COMMENTS::
///FOOTER:
%%DEFAULT::
```

The in-page schematic contains all the basic elements: header, main content, and a footer.

If the **Post Format** option is filled in with this schematic, a sidebar will ***still*** show if the **Sidebar Format** option contains data.

However, if the individual post has the schematic above i.e. contained within the page meta data, the sidebar ***will not*** be shown.

The elements: "///HEADER:","///MAIN:" and "///FOOTER:" are schematic tags. They need to be present for an alternative schematic to be processed. The Main Area is the main content area of a post or page. It is also where you would have a sidebar if present.

Default Page Content

The default page schematic is:

```
%%CONTENT_META::
%%CONTENT_DEFAULT::
%%COMMENTS::
```

Pagination for pages only appears if you have used the "<!—nextpage—>" tag in your WordPress content. Pages are not linked together like posts. If you added "% %PAGINATION::"only the linked pages pagination would display!

Default Footer Content

The default footer schematic is:

```
%%MENU::
mlk={href="SITE_HOME" name="Home" src="IMAGES/
home_icon.png" class="home-ref"}:
%%SEARCHBAR::
```

This will show a single home icon in the bottom left of the footer, which has a search icon below it. Clicking on the icon shows the search bar input field.

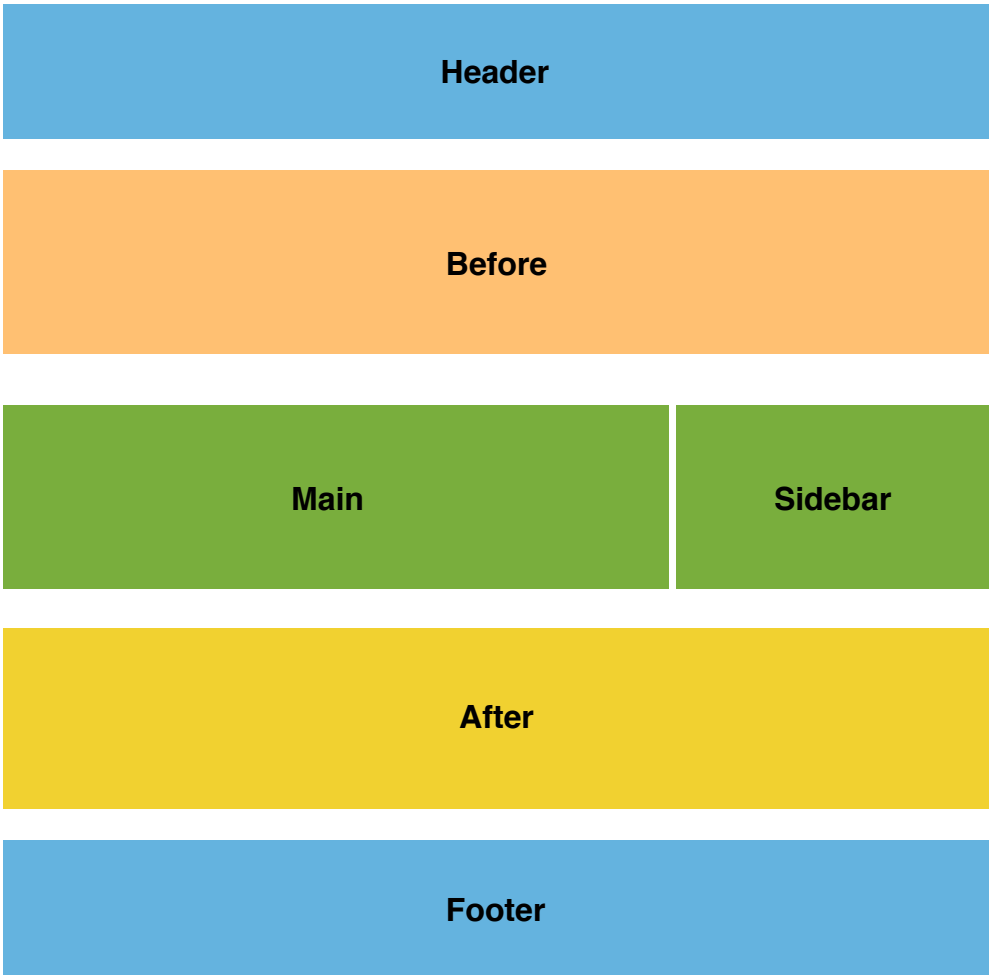
For quick access to Footer Widgets you can use:

%%WIDGETS::*<1 to 4>*

This will allow any widgets you put in any of the 4 Widget Areas in the WordPress Admin to be shown.

Other format fields

There are two additional format fields in PoliMorf: **Before Main Format** and **After Main Format**. These formats are used to add data above and below the Main Area (which may also include a sidebar). Pictorially the format is laid out like so:



Remember that the Main Area can be full-width with no Sidebar if **Sidebar Format** is empty or the in-page schematic does not include the "///SIDEBAR:" tag.

To include the extra tags you can do one of two things:

- Fill in the options in **Before Format** and **After Format** in *PoliMorf Settings->Styling*.
- Add "///BEFORE:" and "///AFTER:" tags with commands or a default reference.

For example, a default page schematic could be extended to:

```
///HEADER:
%%DEFAULT::
///BEFORE:
%%DEFAULT::
///MAIN:
%%CONTENT_META::
%%CONTENT_DEFAULT::
%%COMMENTS::
///AFTER:
%%DEFAULT:
///FOOTER:
%%DEFAULT::
```

3

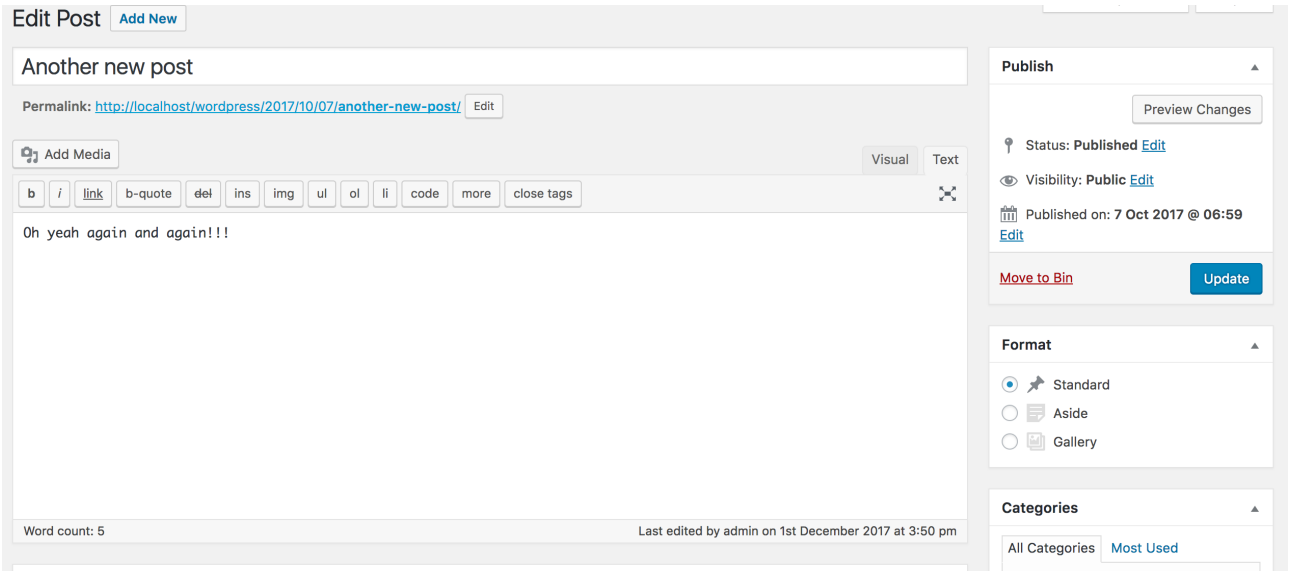
Adding & Changing Content

Using commands to add different types of content

From the default post schematic format you may have noticed the command for adding the default WordPress content of a post or page:

%%CONTENT_DEFAULT::

When typically constructing a post or page you use the WYSIWYG (What You See Is What You Get) editor i.e.



The content you edit in this box is what comes up on screen. You can choose to see what it looks like visually i.e. like a document; or in code (Text) i.e. with html tags.

If you choose to only use WordPress, the default page and post schematic format will show the content using the above command. This content is that which is processed by WordPress itself and so it will allow a wide range of html tags and attributes as per the WordPress rules (The WordPress Codex).

Extra Content

If you wish to add content above and below whatever you edit here you can use the %%CONTENT:: command. It takes one sub-command, 'TEXT':

```
%%CONTENT::
TEXT=[    ]:
```

We already came across this command structure when adding extra text to the Footer. It works the same in the Main Area.

The TEXT command allows the user to add html by using '[ ]' tags instead of '<>' html tags. For example a paragraph is processed from '[p]' to '<p>'. This ability to write in non-html tags means that TEXT can be used in fields that would not allow html tags. It also adds in a layer of safety as can only use the '[ ]' tags available.

This becomes extra important when using remote schematics. In the same way that shortcodes in WordPress get replaced with functionality, '[ ]' html tags get replaced.

The exceptions to this are when you wish to add images, links and image links (images with a link). To add an image you use the image keyword command i.e.

```
img={ }:
```

For links:

```
link={}:
```

And for links with images ie. having html of the form <img.../>:

```
img_link={}:
```

The attributes are those allowed by HTML5 for images and general attributes except for ‘data-*’ types. These are custom data attributes and as such if you wish to add these do so using the WordPress WYSIWYG editor.

The available attributes for images and links are:

General Attributes	Image Attributes	Link Attributes
accesskey	fa	text
class	alt	charset
contenteditable	crossorigin	coords
contextmenu	height	download
dir	ismap	href
draggable	longdesc	hreflang
dropzone	sizes	media
hidden	src	name
id	srcset	rel
lang	usemap	rev
spellcheck	width	shape
style		target
tablindex		type
title		
translate		

The link attribute “text” is not an HTML5 attribute but is used when creating a link. It is the text that normally sits between the <a>text html tags. If no text is specified then the link will not be shown. The image attribute “fa” is the Font Awesome class in an icon link.

<i class=“fa ….. “ aria-hidden=“true”></i>

If you want to use an icon put the class details as a “fa” attribute and the icon will be output. The icon will appear **in addition** to any image.

Links and images can be added anywhere within the TEXT=[ ]: syntax, though preferably within paragraphs. For example, to add an image called “image-1.gif” that was uploaded to your site through WordPress in October 2017, you could use:

```
TEXT=[
[p]img={ src="MEDIA/2017/10/image-1.gif" class="image-gif"
alt="My Image" }:[/p]
]:
```

This will translate to html as:

```
...
<p>
<img src="http://yoursite/wp-content/uploads/2017/10/image-1.gif"
class="image-gif" alt="My Image" />
</p>
...
```

The dots represent the surrounding divs added by PoliMorf.

To add a link, you could do the following:

```
TEXT=[
[p]This is a link={ href="MEDIA/2017/10/image-1.gif" class="image-
gif"" name="My Image" text="link" target="_new"}: to a new image.[/
p]
]:
```

This will translate to html as:

```
...
<p>
This is a <a href="http://yoursite/wp-content/uploads/2017/10/
image-1.gif class="image-gif" name="My Image"
target="_new>link</a> to a new image.
</p>
...
```

To add an image link, you could do the following:

```
TEXT=[
```

```
[p]img_link={ href="MEDIA/2017/10/image-1.gif"  src="MEDIA/
2017/10/image-1.gif" class="image-gif"" alt='My Image" name="My
Image" text="link" target="_new"}:[/p]
]:
```

This will translate to html as:

```
...
<p>
<a href="http://yoursite/wp-content/uploads/2017/10/image-1.gif"
class="image-gif" name="My Image" target="_new">
<img src="http://yoursite/wp-content/uploads/2017/10/image-1.gif"
alt="My Image" /></a>
</p>
...
```

The general html attributes will be associated with the link rather than the image.

Meta Data

For posts and pages, the other important command was:

```
%%CONTENT_META::
```

This command has arguments like %%NAV:: in that it uses the ={}: identifiers. If you have no arguments, the default action is dependent on whether it is a post or a page.

- For posts, the title, author (linked), publish date (linked), categories (linked) and tags (linked) are shown.
- For pages, the title, author (not linked), and publish date (not linked) are shown.

Post meta data is linked as it leads to specific template pages listing the particular meta data. These are "Author" or "Category" pages and are core WordPress template pages. So clicking on the author name will produce a list of posts by the author.

Pages are standalone items and so PoliMorf does not link to them like posts. If you wanted to have this functionality you could code a special function for it, which would be beyond this user guide.

To hide any of these items you need to list all the rest except the one you want to hide i.e.

```
%%CONTENT_META::
title={}:
author={}:
date={}:
cat={}:
```

This will hide the tags on a post. Hiding tags will not have any effect on a page, though hiding the author will.

Just as a reminder then, the command:

```
%%CONTENT_META::
title={}:
author={}:
date={}:
cat={}:
tag={}:
```

Is equivalent to:

```
%%CONTENT_META::
```

Inserting Content into WordPress default content

As PoliMorf does not do everything possible with content formatting, you can use WordPress in conjunction with schematic content commands.

This is achieved using the same idea as with shortcodes! PoliMorf allows you to define insert tags that can be placed in WordPress content. It processes commands associated with these tags into html outputs and then replace the insert tags in the content with html. It uses the content filters in WordPress to do this, hence you can even insert shortcodes if you like!

Note: you would still need to check the priority of when shortcodes are processed. The content filtering described here will occur when WordPress can do it. No priority is defined. Shortcodes on the other hand may have a priority, typically after the main content is rendered so that the code is present.

As an example, say you had some content in a Wordpress page such as this (taken from the default sample page in WordPress):

> This is an example page. It's different from a blog post because it will stay in one place and will show up in your site navigation (in most themes). Most people start with an About page that introduces them to potential site visitors. It might say something like this:
>
> Hi there! I'm a bike messenger by day, aspiring actor by night, and this is my website. I live in Los Angeles, have a great dog named Jack, and I like piña coladas. (And gettin' caught in the rain.)
>
> …or something like this:

You might want to insert an image above this so you would add the following:

> [[Insert_1]]
>
> This is an example page. It's different from a blog post because it will stay in one place and will show up in your site navigation (in most themes). Most people start with an About page that introduces them to potential site visitors. It might say something like this:
>
> Hi there! I'm a bike messenger by day, aspiring actor by night, and this is my website. I live in Los Angeles, have a great dog named Jack, and I like piña coladas. (And gettin' caught in the rain.)

…or something like this:

Using the tag “[[Insert_1]]” you would leave a reference point in the WordPress content. Then in the schematic for this entry (a page) you could write the following:

```
///HEADER:
%%DEFAULT::
///MAIN:
%%CONTENT_META::
%%CONTENT_INSERT::
[[Insert_1]]
TEXT=[
[p]img={ src=“link-to-image.gif” alt=“Image inserted at Insert_1”
name=“image-1.gif”}:[/p]
]:
%%COMMENTS::
///FOOTER:
%%DEFAULT::
```

The important change is the command highlighted in red.

It does not say “%%CONTENT_DEFAULT::” anymore but instead it uses the command “%%CONTENT_INSERT::”. This command still uses the default WordPress content, but it allows the user to enter extra text content.

When the new content is processed it will include an image above the original content.

Now, you may have noticed that in this example you could have written the whole thing as %%CONTENT:: with TEXT=[ ]:. So you didn’t need to use the WordPress editor.

But what if you wanted to include links to videos or audio streams? Currently PoliMorf does not have specific commands for this as often you need to have a specific plugin on your site if you are not just linking from somewhere like YouTube or SoundCloud. You can add explicit code in an indirect way using “[tag]” and “[/

tag]” text replacements (“<“ and “>”) but as a more simple way you can combine WordPress content with inserts.

.

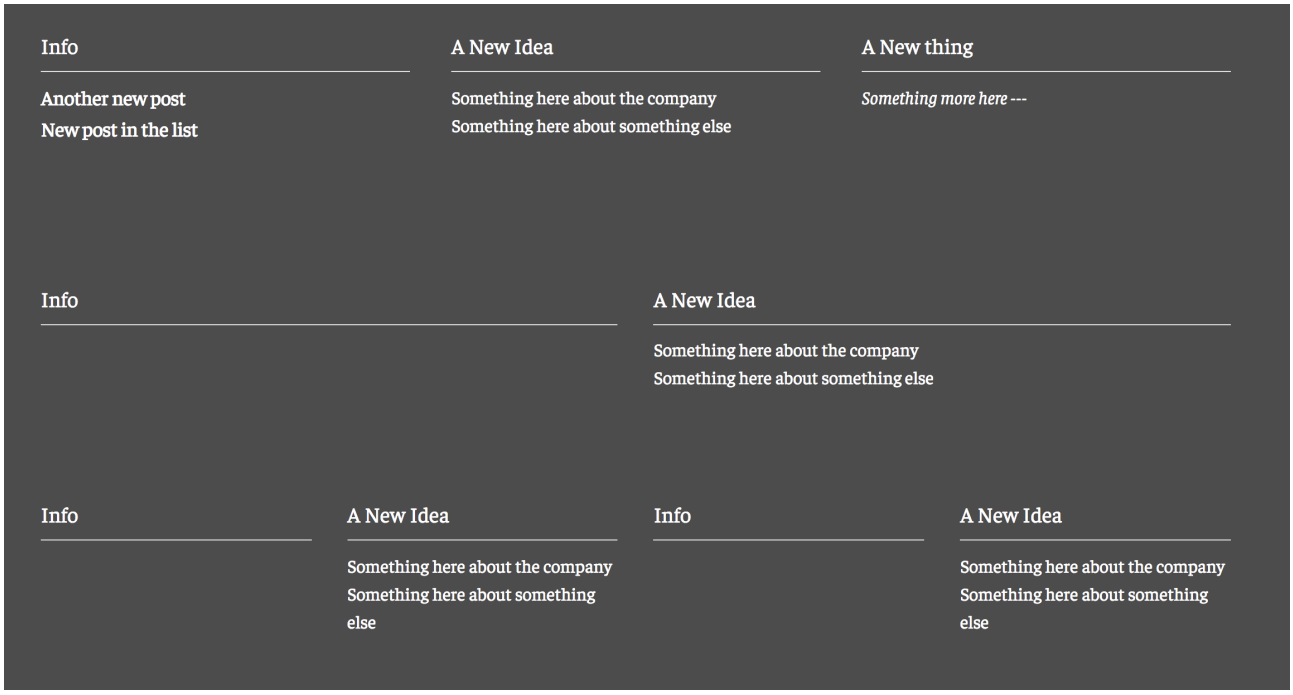
4
Adding Boxes
Using Tabular-like Elements

At times adding box elements (or 'grid elements') that are not the same as html tables, gives an extra dimension to your site. You can add columnar information instead of linear lists of block text. There are many ways to do this, but to make it easier PoliMorf has a set of commands called N-Boxes. N stands for 2, 3 or 4.

The commands are:

%%2-BOX:: BOX1: *..commands* BOX2: *..commands*	%%3-BOX:: BOX1: *..commands* BOX2: *..commands* BOX3: *..commands*	%%4-BOX:: BOX1: *..commands* BOX2: *..commands* BOX3: *..commands* BOX4: *..commands*

Below is an example of a 3, 2 and 4-box element placed one after another with various menu, text and titles.



You can use N-Boxes anywhere, in the Header, Main elements and the Footer.

Each BOX allows a number of subcommands:

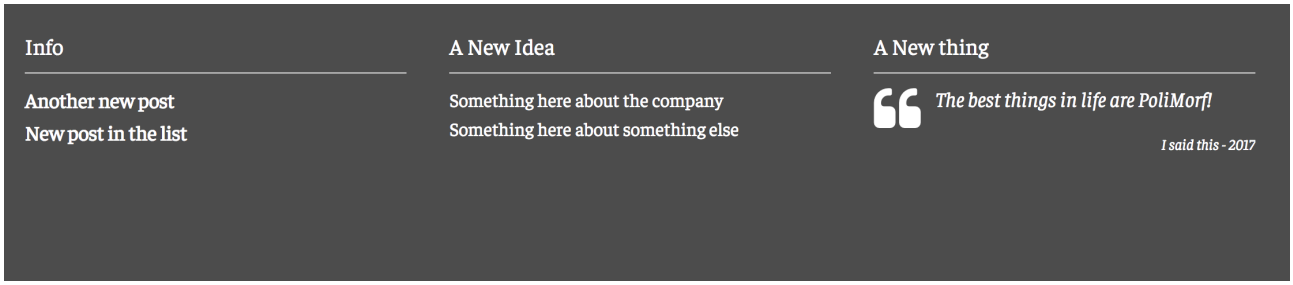
- TEXT=[ ]:
  - You have already seen how this works
- TITLE=[<title name>]:
  - The syntax is simply the name of the title e.g. TITLE=[A New Idea]:. You can add spaces before and after the title as it will be trimmed.
- QUOTE=[body={ text allowing for [] tags }: ref={ text allowing for [] tags }: link={ href=... }:]:
  - This is a lower level version of the %%QUOTE:: command where there is a quote body and a quote reference. The result is a quote with a large (“) start and text aligned to the left then a reference aligned right that can have a link specified by “link”.
- SEARCHBAR=[]:
  - Inserts a searcher - no syntax needed.
- WIDGETS=[]:

- Inserts widgets from areas 1 to 5 (depending on settings). If blank will inserts widgets from Widget Area 1.

- MENU=[mlk={..}: title={..}: wpmenu={...}:]:
  - Menu allows you to add a WordPress menu or your own links. It works like a NAV menu (as touched upon earlier and described in more detail later). It does not include the responsive icon and you cannot add a search bar to the menu as you can just add it to the box.

Examples

If we take the example of the 3-box in the footer, i.e.:

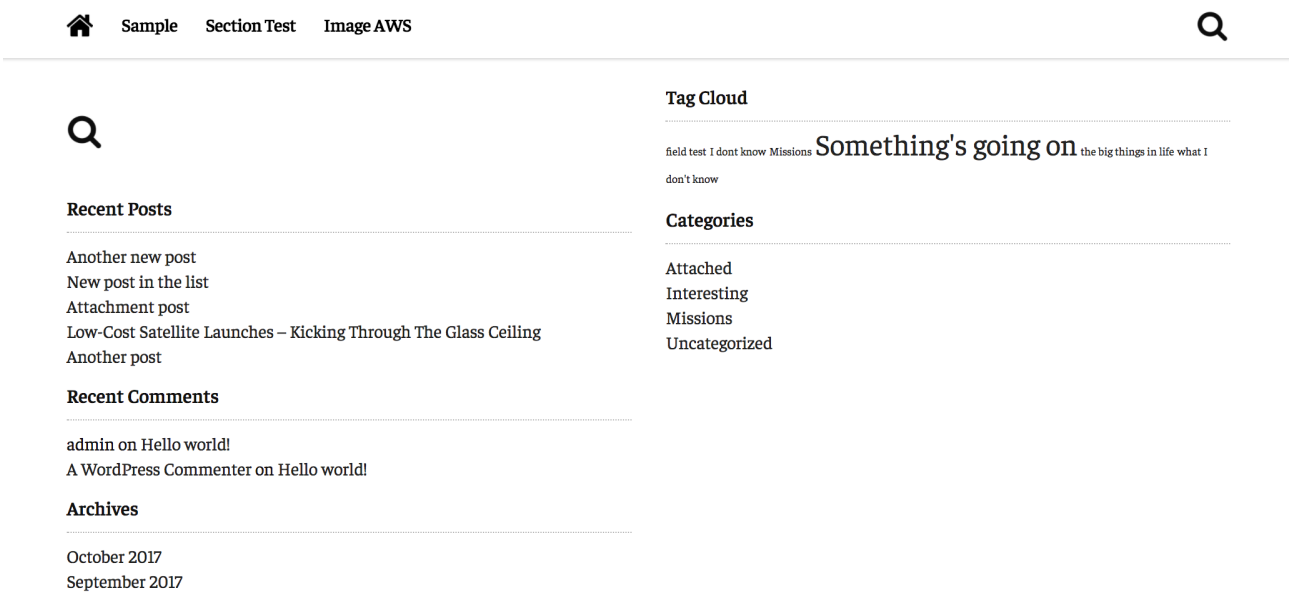


This set of boxes is created as follows:

```
%%3_BOX::
BOX1:
MENU=[
wpmenu={name="Footer Menu" }:
]:
BOX2:
TITLE=[  A New Idea  ]:
TEXT=[
[p]Something here about the company[/p]
[p]Something here about something else[/p]
]:
BOX3:
TITLE=[ A New Thing ]:
QUOTE=[
body={[p]The best things in life are PoliMorf![/p]}:
ref={[p]I said this - 2017[/p]}:]:
```

In the Menus section of WordPress Admin, a menu was created called “Footer Menu” that had two links to posts: “Another new post” and “New post on the list”. This is what is referenced in “wpmenu={}:”

An example of widgets applied to a 2-box element is shown below:



This is created using the following box command:

```
%%2_BOX::
BOX1:
WIDGETS=[]:
BOX2:
WIDGETS=[2]:
```

Widget Area 1 contains WordPress Widgets: Search, Recent Posts, Recent Comments and Archives. Widget Area 2 contains WordPress widgets Tag Cloud and Categories.

5
More Elements To Play With
Quotes, Menus, Sections and more

Apart from content and boxes, quotes and menus can be added. These elements can also be bundled into Sections.

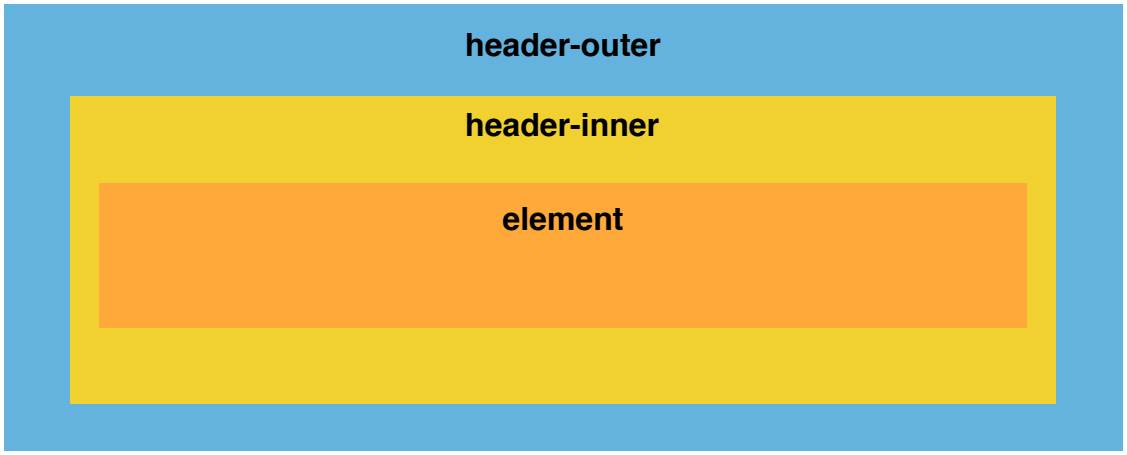
PoliMorf Substructure

PoliMorf is designed to use full-width styling by default. This means having no sidebar. Elements are then arranged as a set of “stripes” where in a particular element you have an outer and inner div then the div associated with the element.

i.e.

        ....

For the header, the classes would be “header-outer” and “header-inner” where the outer has no margins and spreads across the screen. The inner is responsively styled so that it maintains a certain fixed width across the screen.



Schematically this can be shown as above. In reality the divs are the same height.

Sometimes, however you may want to group elements together and have a div to wrap them together. This is achieved using Sections.

In a Section you could have a full-width quote, a 3-box and a section of text all with a background image covering the elements.

Creating Sections

To call a Section you need to use a SECTION command i.e.

```
%%SECTION::
```

As may be apparent, how do you then use schematic commands of the form "%% ...::" to add elements to Sections? The answer is you use a slightly different identifier.

Instead of using "%%" and "::" you use "[%" and ":]" . For example:

```
%%SECTION::
{{ new-section }}
[%2_BOX:]
BOX1:
WIDGETS=[]:
BOX2:
```

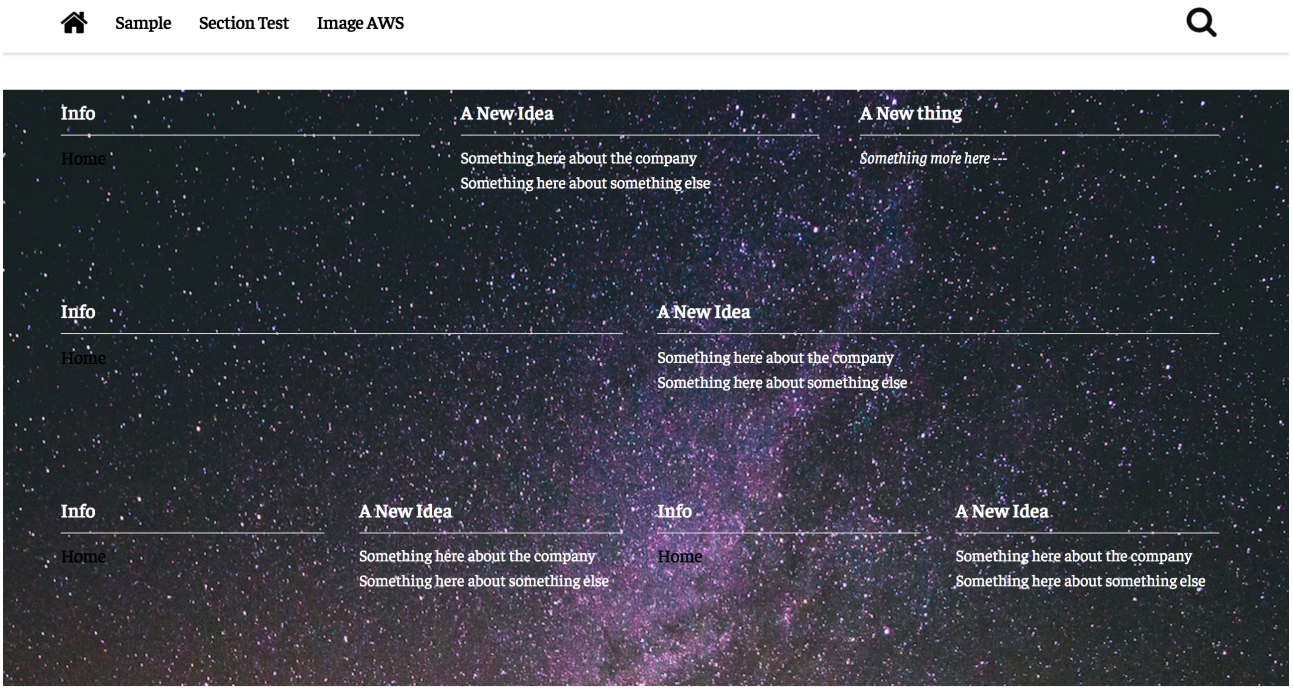
WIDGETS=[2]:
[%QUOTE:]
body={[p]The best things in life are PoliMorf![/p]}:
ref={[p]I said this - 2017[/p]}:

This will add a Section with an id of “new-section” where there are two elements: a 2-Box with widgets, and a quote.

Important: You must provide an id otherwise the Section will not appear.

Since Sections will be unique an id is used rather than a class. You can style the Section using extra css in either the **Manual Styling** option in *PoliMorf Settings->Styling* or in the page/post custom meta field **In-Page Styling**.

The following example shows a series of boxes with a starry background.



This is created by having a Section with a 3,2 and 4 box in it. The background is set by css using styling. It can be set using the “%%BACKGROUND_IMAGE::” addition command. In Chapter 7 you will find out how to set the background to a secure remote image on the Cloud.

Adding Quotes

You should be familiar with adding Quotes as they have been included in various examples already. This section will describe the full syntax and use of a Quote.

Quotes are added at two levels:

- The schematic level using %%QUOTE::
- The subcommand level using QUOTE=[ ]:

Much like with NAV, the QUOTE command uses keywords with the ={}: identifiers :

- body={}:
- ref={}:
- link={ href="" ...<link attributes as in Chapter 3 excluding "text"> }:

When you have a quote you typically have the main text then some attribution reference, hence the use of "body" and "ref". If you wished to link the reference use "link". The commands "body" and "ref" will accept "[ ]" html substitutes in the same manner as occurs with TEXT=[ ]: commands covered in chapter about adding content.

The following examples provide the same output but one is used within a box and the other is used a page level:

1)

```
%%QUOTE::
body={
A lot of time goes by and you learn something or think you do. Then you realise it was all a waste for most of it and that it's the grind that counts.
}:
ref={Bill the Great, 2016, At London Stage School }:
link={href="SITE_HOME"}:
```

2)

```
...
QUOTE=[
body={
A lot of time goes by and you learn something or think you do. Then you realise it was all a waste for most of it and that it's the grind that counts.
}:
ref={Bill the Great, 2016, At London Stage School }:
link={href="SITE_HOME"}:
]:
...
```

Adding Menus

Menus can be added manually using the MENU command. There are 4 keywords that can be summarised as:

- mlk={ ..}:
  - attributes have been described elsewhere
- title={ <title>}:
  - Title as a text string
- wpmenu={...}:
  - attributes have been described elsewhere
- searchbar={}:
  - takes no attributes

If you wanted to add a menu before content you can use:

```
%%MENU::
title={ WP Menu }:
wpmenu={}:
```

This would add a linear menu like so:

Post Lists

If you want to add a list of specific posts and pages that will appear like the list on an Archive page or the Index Page (Blog Front Page) you can use the POST_LIST command. It is not just for posts even though the command refers to posts; it will also display pages.

A Post List can be displayed on its own or within a box or sidebar. It can be called at the schematic command level and sub-command level.

It takes a single keyword of "entry={ }:" where the name is the **last part** of the post or page slug. For example, if you have a postcalled "Another Post About Dogs" and the slug is "http://yoursite.com/2017/10/posts/another-post-about-dogs/" you would use the following syntax:

```
entry={ another-post-about-dogs }:
```

To extend this, say you had 4 posts about dogs including this one, and a page about dog training. A Post List would be as follows.

```
%%POST_LIST::
entry={ another-post-about-dogs }:
entry={ another-post-about-dogs-2 }:
entry={ another-post-about-dogs-3 }:
entry={ another-post-about-dogs-4 }:
entry={ dog-training-101 }:
```

Page and Post Custom Meta

There are descriptions within the Custom Page/Post Settings

Alternative Featured Image url

You can add an alternative **Featured Image** url for the image thumbnail that will appear in post lists for a given page or post i.e in archive pages or in a blog roll. If you don't specify this or a Featured Image, a default image will be shown. You can use constants such as "SITE_HOME" and "MEDIA" in this field.

Custom Excerpt

The **Excerpt** field of a post or page is displayed by clicking on the *Screen Options* menu at the top of the page/post when editing. This is an in-built WordPress menu. Once activated there will be an Excerpt section where you can add some text and html. You can also use the TEXT=[ ]: subcommand here. This excerpt will be displayed in page or post lists. If not it will default to displaying the initial lines of any content you have. Note: this is content what you enter into the WordPress editor not content you set by using formatting or remote data.

Additions

These are commands that are processed in the Header Format only. The result is that they hook various elements and calls to "wp_head" and to "wp_footer". They are used to attach references to stylesheets, fonts and to add effects.

There are 6 commands:

- FONT - a Google font reference
- EFFECTS - Scroll reveal effect
- STYLESHEET - external css ref (can be on AWS)
- BACKGROUND_IMAGE - external background image link (can be on AWS)
- HEADER_CONTENT - you can add script here using text replacements (similar to CONTENT). You can also add a script reference i.e. *<script src="…"></script>*
- FOOTER_CONTENT - you can add script here using text replacements (similar to CONTENT). You can also add a script reference i.e. *<script src="…"></script>*

Font references

To add a font reference (usually a Google font) you only need to add the href reference after the "%%FONT::" command. There is no keyword or attribute i.e:

%%FONT::
<href font reference>

Scroll Reveal effects

You can add a reveal effect (https://scrollrevealjs.org/) using schematic commands. This functionality uses the Scroll Reveal javascript library that you can reference from the link. Reveal effects are a popular way to format sections of your website. They can be used in conjunction with custom classes very easily.

To add a reveal effect you need to call the schematic command "%%EFFECTS::" in the Header. Calling this schematic command anywhere else will not initiate the reveal.

The command for a reveal effect is handled using a keyword "reveal". It works much like elements of the "%%NAV::" command. To make a section of a page animate with a reveal, add the following:

%%EFFECTS::
reveal={ class="<class you want to animate>" settings="<list of settings>" }:

You can call multiple reveal keywords and animate a number of classes differently. For a custom class, remember that the class is applied to the outer and inner wrappers so you may want to specify exactly which part of the structure you want to animate.

For example, for a custom class called "information-box" called on a schematic command: "%%CONTENT_CUSTOM: information-box ::", you would animate the internal content using ".post.information-box". You could animate the whole thing using ".<outer-class>.information-box".

Settings are specific to the Scroll Reveal library. They handle aspects like duration and type of reveal. If you do not specify settings, the default will be used. If you use settings, separate them using commas. For example, to animate the post content of class “information-box” using defaults:

```
%%EFFECTS::
reveal={ class=“.post.information-box” }:
```

To animate the post content of class “information-box” using a duration of 700 ms:

```
%%EFFECTS::
reveal={ class=“.post.information-box” settings=“duration: 700” }:
```

The full settings syntax can be found here - https://github.com/jlmakes/scrollreveal

Stylesheet references

To add a reference to an extra stylesheet you only need to add the url reference after the “%%STYLESHEET::” command using the keyword “ref”:

```
%%STYLESHEET::
ref={ url=“<href font reference>” }:
```

It is also possible to link to a stylesheet stored remotely on the Cloud. Refer to chapter 7. You can add as many keyword calls as you like under the same command.

Background image link references

To add a reference to a background image for a section or class element you only need to add the css name of the element (including the # or .) and the url reference after the “%%BACKGROUND_IMAGE::” command, using keyword “element”:

```
%%BACKGROUND_IMAGE::
```

element={ name="<# or .><name of element>" url="<href font reference>" }:

It is also possible to link to an image stored remotely on the Cloud. Refer to chapter 7. You can add as many keyword calls as you like under the same command.

Header and Footer Content

If you need to add explicit header content, use the command "% %HEADER_CONTENT::". There are two subcommands: TEXT and SCRIPT. TEXT uses the same substitutions as those used for CONTENT, while script has one main attribute (src). You can add as many subcommand calls as you like.

The following schematics achieve the same effect. For TEXT you can also include JavaScript text as it be processed as is. Note that any html tags openers and closers (< and >) will be removed so you still need to add substitutions.

```
%%HEADER_CONTENT::
TEXT=[
[script_open] src="script_reference.js"[/tag][/script]
]:

%%HEADER_CONTENT::
SRC=[ src="script_reference.js" ]:
```

The result will be the following html applied to the end of the information in 'wp_head':

<script src="script_reference.js"></script>

The same subcommands apply for "%%FOOTER_CONTENT::". Content is added to the end of information in "wp_footer".

To change where the information is hooked in "wp_head" and "wp_footer", set the priority values to smaller numbers. The default priority is 510.

In Chapter 7 you can learn how to add a remote script reference.

6

Custom Classes, Fonts, Effects and Styles

How to add styling classes to top-level elements & How to add font, effects and style references to the header

When adding an element by schematic command, you may want it to be unique. To do this you can add a Custom Class when calling the particular element in question. For Sections you may not need to do this as they had a unique ID, but if you have a lot of 3-box elements for example, you may want to be able to style or select them easily.

To add a custom class you append the command with "_CUSTOM:". For example, with a 3-box element:

```
%%3_BOX_CUSTOM: 3-box-unique-class ::
```

You can have spaces before and after the class name if you like. The class name will be trimmed. What this does is apply the class to the top level 3-box div as well as the wrappers.

PoliMorf wraps elements with the Header, Main and Footer sections with an outer and inner class. So adding a Custom Class will do the following behaviour in html:

```
<div class="outer custom-class">
    <div class="inner custom-class">
        <div class="element-class custom-class">
        ....
```

```
        </div>
    </div>
</div>
```

If you try an apply Custom Classes to elements within a larger element, such as a menu within a box, the class will be ignored. Classes can be added to images and links as has been shown previously. All other elements can be styled using the higher level Custom Class call.

Preset Custom Classes

Currently there are a number of preset custom classes

.block-menu

Turns a linear Menu into a vertical list

.home-ref

Inverts a black icon to a white icon. Used for the home icon in the footer.

7
Remote Data Calls
How to leverage the Cloud to enhance your content

One of the primary functions of PoliMorf is to allow the administrator flexibility as to where the content is stored. This is why it uses schematic commands so that a degree of separation between in-page content and structure can be created.

With this in mind, a logical step would be to consider the possibility of storing content and schematics somewhere else. These could then be referenced locally with simple commands which in turn would reduce your data footprint on your hosting service.

Why is this important? The primary reason would be cost.

PoliMorf is configured to work with Amazon Web Services (AWS) at the moment. Getting a file from Amazon costs much less than 1 US cent. Getting a file from your hosting services may be considerably more. Add to this the improvement in the API to retrieve data or create temporary links to data in storage, and that this can be achieved *securely*, we now have the ability to leverage this cost differential in our sites. We can easily set up secure fetching of data without complicated overhead or third-party plugins.

PoliMorf achieves this functionality by retrieving remote schematic data and processing it; or by creating temporary pre-signed urls to data in the Cloud.

The idea of using pre-signed temporary links is not new but it has become a lot easier to do with free third-party software. The idea of hosting schematics is a way for website elements to be parsed by not using direct html or JavaScript. This creates an extra level of security and is consistent with schematic commands you may use locally on your site.

It also means you can edit your site from another place and using simple text tools rather than in the WordPress Admin.

Remote calls at Global and Schematic Level

There are two ways to call remote data: one is at the global schematic level; the other is within a schematic itself. A call at the global schematic level uses one command to retrieve a full schematic and reproduce this on a page. Within this schematic there may be more remote calls, to images or other content.

A remote call within a schematic command list will replace itself with the content of the fetched schematic. This is then processed as if the commands were in the list already. For example, if a remote schematic had 3 schematic commands in it:

```
%%COMMAND_1::
%%COMMAND_2::
%%CALL_TO_REMOTE_SCHEMATIC::
```

Would become

```
%%COMMAND_1::
%%COMMAND_2::
%%REMOTE_COMMAND_1:
%%REMOTE_COMMAND_2:
%%REMOTE_COMMAND_3:
```

Remote calls can also be made for images and links, where the href/src/url attribute is replaced by a secure pre-signed link with a specified expiry time. For example, you could choose to reference an image in a post with one stored in the cloud. You would use an img={}: command within a TEXT=[ ]: command and configure the attributes to refer to the remote data. All other html attributes for the image would be as normal. The result would be an src value that is a pre-signed link.

Configuring Remote Assets

To call remote data you first need to set up an account on Amazon Web Services (AWS). In previous iterations it was possible to set up a master account (a root level account) and use this for setting permissions to data. Things have moved on these days and now you use a service called Identity Access Management (IAM).

Control over the files and folders at an administrative level is still controlled by a root user, which is the account you create when you sign up. However, IAM provides a layer removed from the root user so that you never need to share the root user credentials with third parties. You create a specific IAM user instead.

The idea of using AWS with PoliMorf is that you store your files in an S3 bucket. S3 is an AWS service allowing remote storage and a bucket is a type of storage location that can contain folders as well as files. You can access buckets directly as a root user or using policies that allow certain IAM users differing types of access.

You can learn a lot more about policies and IAM on the AWS site, but to simplify the process, this section will deal with setting a Read-Only-Access policy that you would have for your site. This policy would prevent someone altering data as they would not have Write-Access.

The first thing is to create an IAM user from your root account. AWS has instructions on how to do this. Once created you need to go to your root level dashboard in IAM and assign a policy to this user. Then you go back to S3 and assign a similar policy to the bucket you want to access. Once this is done, the security credentials for the user, the public and private keys, can be used to gain access to this bucket.

User Policy

The best practice is to assign a user to a Group and then apply a Group policy. In the IAM dashboard you create a Group, for example “SiteAccess”. You then add a user to the Group.

Once this is done, click on the Permissions tab and assign a new policy. For bucket “polimorfsite” it will be as follows:

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::polimorfsite/*",
            "Condition": {}
        }
    ]
}
```

Bucket Policy

The bucket policy is applied under the Permissions tab for a particular bucket in the S3 console (**Note: Not IAM**). You can use the policy generator to get the policy skeleton correct. The policy for allowing Read-Only access for user “SiteUser1” to bucket “polimorfsite” on account number 12345 (the root account number) is as follows:

```
{
    "Version": "2012-10-17",
    "Id": "S3-Account-Permissions",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::12345:user/siteuser1"
            },
            "Action": [
```

.

```
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::polimorfsite",
                "arn:aws:s3:::polimorfsite/*"
            ]
        }
    ]
}
```

Once this is set-up any remote calls to the bucket will work.

PoliMorf Settings->Cloud

There are only two settings that need to be set up in PoliMorf: **Public Key** and **Private Key**.

When you enter the value of **Public Key** it will stay visible after saving. The value of **Private Key** is transformed into a set of dots and will not be visible to the user. This is a security feature.

You do not store your region, bucket name, file name or timeout values in the options. You simply set these explicitly in the remote calls.

Remote Syntax

Remote calls are made using a variety of commands ranging from a complete schematic call to individual links and image references.

Top Level Remote Call

To call a complete schematic you would set the Page Format for a specific page or post. You would use the of schematic level tag “///REMOTE:” followed by one command to a remote schematic i.e.

```
///REMOTE:
%%REMOTE_SCHEMATIC_AWS::
SCHEMATIC=[ filename=“schematic-for-page.txt” region=“us-east-1”
bucketname=“polimorfsite” ]:
```

Note that you do not need to use the attribute “timeout” for schematic calls. Region name syntax is described in AWS documentation and can be easily searched for.

Schematic command level call

To call remote schematic data, you do the same thing as with top-level call but without the “///REMOTE:” tag. For example, if you just wanted the Main section to be accessed remotely:

```
///HEADER:
%%DEFAULT::
///MAIN:
%%REMOTE_SCHEMATIC_AWS::
SCHEMATIC=[ filename=“schematic-for-page.txt” region=“us-east-1”
bucketname=“polimorfsite” ]:
///FOOTER:
%%DEFAULT::
```

Remote link and image calls

You incorporate remote calls in links and images using the same attributes. You can also include “timeout” if you want to override the default value of 300 seconds. You do not need to use href/src/url attributes as these are created from the remote information.

For a link to a document stored you could have:

```
...
link={ href="document.pdf" name="Linked Document" text="Link
here!" target="_new" }:
...
```

For a link to the same document stored remotely in region US East 1, in bucket "mysite" you would have:

```
...
link={ filename="document.pdf" region="us-east-1" bucket="mysite"
name="Linked Document" text="Link here!" target="_new" }:
...
```

This would create a temporary link to the document with an expiry time of 300 seconds. Adding 'timeout="500"' would change the expiry time to 500 seconds. The default limits on timeout are 5 seconds to 86400 seconds (1 day).

For a link to an image stored locally you could have:

```
...
img={ src="image1.gif" name="Linked Image" alt="Linked image" }:
...
```

For a link to the same image stored remotely you would have:

```
...
img={ filename="image1.gif" region="us-east-1" bucket="mysite"
name="Linked Image" alt="Linked image" }:
...
```

For a local link with an image stored remotely you use a specific set of remote references with a prefix of "image_":

...
img_link={image_filename="image1.gif" image_region="us-east-1" image_bucket="mysite" name="Linked Image" alt="Linked image" href="http://locallink" }:
...

For a remote link with an image stored remotely you use a specific set of remote references with a prefix of "image_" and one with "link_". This may be a link to a download or a document:

...
img_link={image_filename="image1.gif" image_region="us-east-1" image_bucket="mysite" name="Linked Image" alt="Linked image" link_filename="download.pdf" link_region="us-east-1" link_bucket="mysite"}:
...

You can also use additional HTML5 attributes as discussed previously.

Alternative Featured Image Url remote call

You can add a remote call in the **Alternative Featured Image Url** field in the Custom Page/Post Settings section. Instead of using a url you can specify a file on the Cloud (AWS). The "timeout" attribute is optional (signified by <> brackets):

REMOTE_IMAGE_AWS=[filename="background.gif" region="us-east-1" bucket="mysite" <timeout="200">]:

Custom Excerpt remote call

You can add a remote call in the **Excerpt** field of a post or page. Instead of using a text or the TEXT subcommand, you can specify a file on the Cloud (AWS). The "timeout" attribute is optional (signified by <> brackets):

REMOTE_SCHEMATIC_AWS=[filename="excerpt.txt" region="us-east-1" bucket="mysite"]:

Remote BACKGROUND call

In Sections, you can add a remote link to a background image using the BACKGROUND command. To use this command you write the following syntax within a correctly formatted SECTION command.

```
[— BACKGROUND=[ filename="background.gif" region="us-east-1"
bucket="mysite" <timeout="200"> ]: —]
```

The command has opening and closing identifiers "[—" and "—]". These are used in processing to remove the command and replace it with an html style tag, within which is a css link to the background file. If the identifiers are not present the BACKGROUND command is ignored.

Inline Styling and Remote Stylesheet Reference

You can also add styling within the remote file. This is useful if you want to include remote styling in a schematic. The STYLE command has no subcommands or keywords. It is called at the schematic command level. Any html will be stripped from the syntax so make sure that it is only css.

```
%%STYLE::
<css styling syntax...>
```

For a more generic approach, you could create a separate stylesheet and store it on the Cloud. You can reference this sheet using the addition command "% %STYLESHEET::" placed in the Header. Instead of using "url" you use the remote settings similar to those in the command "BACKGROUND" above.

```
%%STYLESHEET::
ref={
filename="<stylesheet.css>" region="us-east-1" bucket="mysite"
<timeout="200">
}:
```

Header and Footer Content Script Reference

If you have a lot of script (JavaScript) it is best placing it in a ".js" file and storing it locally or in the Cloud. For a remote reference that is placed in the head of your page use the SCRIPT subcommand with remote attributes e.g:

```
%%HEADER_CONTENT::
SRC=[
filename="<script1.js>" region="us-east-1" bucket="mysite"
<timeout="200">
]:
```

You can call as many SCRIPT instances within one "%%HEADER_CONTENT::" command as you like. Similarly you can do this with "%%FOOTER_CONTENT::".

8
Custom Functions and Attributes

How to create your own functions and customise html attributes

You can add your own php functions in the Custom Functions file. If you want to add functions that will be processed as PoliMorf functions they have to follow a basic format.

PoliMorf Functions

All schematic level commands (%%...::) have similar input parameters. If you want to create your own schematic command function you would follow the same format:

```
function new_polimorf_function($syntax,$custom_class,$placement)
```

The parameter $syntax is whatever comes after your command. $custom_class was covered in the previous chapter. $placement refers to the section in which the function would be used.

There are a number of placement constants that you can use. These are are listed in the file called 'pcom_defaults.php' under *theme/theme_dev/main/functions/pcom/*.

i.e.

Constant	Value
PCOM_HEADER_PLACEMENT	HEADER
PCOM_MAIN_PLACEMENT	MAIN
PCOM_MAIN_WITH_SIDEBAR_PLACEMENT	MAIN_WITH_SIDEBAR
PCOM_WITH_SIDEBAR_PLACEMENT	WITH_SIDEBAR
PCOM_FOOTER_PLACEMENT	FOOTER

Header, Main and Footer placements should be self-explanatory by now. Main also covers After and Before placements as they do not have sidebars. When a sidebar is present, Main With Sidebar refers to the main section and With Sidebar refers to the Sidebar section.

If you wanted your function to only appear in the Footer it would use the Footer placement constant as part of any logic condition.

Linking custom functions to schematic command processing

In order to let PoliMorf process a custom schematic function, you need to define a command in the global command function list and give it a callback reference

i.e.

> *$GLOBALS['command_functions_list']['YOUR_COMMAND'] = 'function_callback'*

Essentially what you are doing is adding a new element to the list. You can do the same thing if you create a new addition function.

> *$GLOBALS['addition_functions_list']['YOUR_COMMAND'] = 'function_callback'*

Useful PoliMorf functions

There are two useful PoliMorf functions that will help with parsing syntax. These are:

- pcom_get_strings_syntax_separator
- pcom_process_command_open_close_syntax

pcom_get_strings_syntax_separator

This function has there arguments:

pcom_get_strings_syntax_separator($syntax,$separator,$trim) where $syntax is the input string, $separator is the string you wish to split the $syntax at, and $trim is a Boolean that is used to either trim the results or not.

If $trim is True then the resultant strings are trimmed for front and end whitespace.

The output of the function is an array with three members:

- command_found - true or false
- syntax_before - string before separator
- syntax_after - string after separator

If no separator is found, “syntax_before” is set to $syntax.

pcom_process_command_open_close_syntax

This function has two arguments:

pcom_process_command_open_close_syntax($syntax,$args) where $syntax is the input string and $args are the open and close identifiers for the command or keyword you wish to process.

The default $args array is for subcommand identifiers. If you set $args to null the function will look for these identifiers in $syntax. If you wish to use other identifiers make an array in php as follows:

```
$args=array(
‘open’ => <some opening identifier>,
‘close’ => <some closing identifier>);
```

The output of the function is an array with four members:

- command_found - true or false
- command - the command
- command_syntax - the syntax in between the open and close identifiers
- syntax_after - string content after the close identifier

The “command” can be best used for subcommands and keywords but it can be used for schematic commands. In this particular case “command_syntax” will be the command and “syntax_after” the syntax.

Custom Attributes

In the file ‘pcom_globals.php’, there are a series of arrays dealing with mapping custom attributes to mandatory attributes. For example, it allows you to write ‘link_to’ for ‘href’ when creating schematics. The ‘href’ attributes will be output in the html though you can customise the way you design the html structure. It would have a use if you wanted to code in a different language.

The arrays are reproduced below. The arrays are formatted as key-value pairs where the key is the left hand side and the value is the right hand side.

To use a different attribute **make a copy of the complete array and define it in your custom functions file.**

Then change **the left hand side (the key)** or add a new key.

```
$GLOBALS[‘PM_CUSTOM_LINK_ATTRIBUTES’] = array(
  'text' => 'text',
  'charset' => 'charset',
  'coords' => 'coords',
  'download' => 'download',
  'href' => 'href',
```

```
  'hreflang' => 'hreflang',
  'media' => 'media',
  'name' => 'name',
  'rel' => 'rel',
  'rev' => 'rev',
  'shape' => 'shape',
  'target' => 'target',
  'type' => 'type'
);

$GLOBALS[‘PM_CUSTOM_HTML_ATTRIBUTES’] = array(
  'accesskey' => 'accesskey',
  'class' => 'class',
  'contenteditable' => 'contenteditable',
  'contextmenu' => 'contextmenu',
  'dir' => 'dir',
  'draggable' => 'draggable',
  'dropzone' => 'dropzone',
  'hidden' => 'hidden',
  'id' => 'id',
  'lang' => 'lang',
  'spellcheck' => 'spellcheck',
  'style' => 'style',
  'tabindex' => 'tabindex',
  'title' => 'title',
  'translate' => 'translate'
);

$GLOBALS[‘PM_CUSTOM_IMAGE_ATTRIBUTES’] = array(
'alt' => 'alt',
'crossorigin' => ‘crossorigin',
‘fa’ => ‘fa’
'height' => 'height',
'ismap' => 'ismap',
'longdesc' => 'longdesc',
'sizes' => 'sizes',
```

```
'src' => 'src',
'srcset' => 'srcset',
'usemap' => 'usemap',
'width' => 'width'
);

$GLOBALS[‘PM_CUSTOM_REMOTE_ATTRIBUTES’] = array(
'filename' => 'filename',
'region' => 'region',
'timeout' => 'timeout',
'bucketname' => 'bucketname'
);

$GLOBALS[‘PM_CUSTOM_REMOTE_ATTRIBUTES_REMOTE_LINK’] = array(
'link_filename' => 'link_filename',
'link_region' => 'link_region',
'link_timeout' => 'link_timeout',
'link_bucketname' => 'link_bucketname'
);

$GLOBALS[‘PM_CUSTOM_REMOTE_ATTRIBUTES_REMOTE_IMAGE’] = array(
'image_filename' => 'image_filename',
'image_region' => 'image_region',
'image_timeout' => 'image_timeout',
'image_bucketname' => 'image_bucketname'
);

$GLOBALS[‘PM_CUSTOM_MENU_COMMAND_MLK_KEYWORD_ATTRIBUTES’] = array(
'href' => 'href',
'name' => 'name',
'src' => 'src',
'class' => ‘class’,
‘fa’ => ‘fa’
);
```

```
$GLOBALS[‘PM_CUSTOM_WPMENU_COMMAND_ATTRIBUTES’] = array(
'name' => 'name',
'class' => 'class'
);

$GLOBALS[‘PM_CUSTOM_LOGO_ATTRIBUTES’] = array(
'src' => 'src'
);

$GLOBALS[‘PM_CUSTOM_SITE_REPLACEMENTS’] = array(
'SITE_HOME' => 'SITE_HOME',
'IMAGES' => 'IMAGES',
'UPLOADS' => 'UPLOADS'
);

$GLOBALS[‘PM_CUSTOM_REVEAL_EFFECTS_ATTRIBUTES’] = array(
'class' => 'class',
'settings' => 'settings'
);

$GLOBALS[‘PM_CUSTOM_INSERT_STYLESHEET_REF_ATTRIBUTES’] = array(
'url' => 'url'
);

$GLOBALS[‘PM_CUSTOM_INSERT_BACKGROUND_IMAGE_ATTRIBUTES’] = array(
'url' => ‘url’,
‘name’ => ‘name
);

$GLOBALS[‘PM_CUSTOM_HEADER_CONTENT_ATTRIBUTES’] = array(
'src' => ‘src’
);

$GLOBALS[‘PM_CUSTOM_FOOTER_CONTENT_ATTRIBUTES’] = array(
'src' => ‘src’
);
```

Custom syntax

You can also change the default syntax commands, subcommands and keywords by setting them in your custom functions file. All these elements are set as globals. To add your own syntax you would need to define them before they are set as default.

For example, to set the Menu command "%%MENU::" to "% %NAVIGATION_MENU::" instead, you would define the global PM_MENU_COMMAND with the new value:

$GLOBALS['PM_MENU_COMMAND'] = 'NAVIGATION_MENU';

You would define this **in the custom functions file**. The table below details the syntax you can change using this method.

Name i.e. $GLOBALS['<Name>']	Default Value (strings unless otherwise stated)
PM_HEADER_SCHEMATIC_TAG	///HEADER:
PM_BEFORE_MAIN_SCHEMATIC_TAG	///BEFORE:
PM_MAIN_SCHEMATIC_TAG	///MAIN:
PM_SIDEBAR_SCHEMATIC_TAG	///SIDEBAR:
PM_AFTER_MAIN_SCHEMATIC_TAG	///AFTER:
PM_FOOTER_SCHEMATIC_TAG	///FOOTER:
PM_REMOTE_SCHEMATIC_TAG	///REMOTE:
PM_CUSTOM_CLASS_DECLARATION	_CUSTOM:
PM_DEFAULT_COMMAND	DEFAULT
PM_NAV_COMMAND	NAV
PM_MENU_COMMAND	MENU
PM_NAV_MENU_LINK_KEYWORD	mlk
PM_NAV_LOGO_KEYWORD	logo
PM_NAV_SEARCHBAR_KEYWORD	searchbar
PM_WP_MENU_KEYWORD	wpmenu
PM_BOX_COMMAND_ROOT	_BOX
PM_BOX_BOX	BOX

Name i.e. $GLOBALS['<Name>']	Default Value (strings unless otherwise stated)
PM_MAX_NO_BOXES	4 (integer)
PM_TEXT_COMMAND	TEXT
PM_TEXT_ATTRIBUTE	text
PM_TEXT_IMG_KEYWORD	img
PM_TEXT_LINK_KEYWORD	link
PM_TEXT_IMG_LINK_KEYWORD	img_link
PM_FA_ATTRIBUTE	fa
PM_QUOTE_COMMAND	QUOTE
PM_QUOTE_BODY_KEYWORD	body
PM_QUOTE_REF_KEYWORD	ref
PM_QUOTE_LINK_KEYWORD	link
PM_TITLE_COMMAND	TITLE
PM_TITLE_KEYWORD	title
PM_CONTENT_COMMAND	CONTENT
PM_CONTENT_DEFAULT_COMMAND	CONTENT_DEFAULT
PM_CONTENT_WITH_INSERT_COMMAND	CONTENT_INSERT
PM_CONTENT_META_COMMAND	CONTENT_META
PM_CONTENT_META_TITLE_KEYWORD	title
PM_CONTENT_META_AUTHOR_KEYWORD	author
PM_CONTENT_META_DATE_KEYWORD	date
PM_CONTENT_META_CATEGORY_KEYWORD	cat
PM_CONTENT_META_TAG_KEYWORD	tag
PM_POST_LIST_COMMAND	POST_LIST
PM_POST_ENTRY_KEYWORD	entry
PM_SECTION_COMMAND	SECTION
PM_INSERT_BACKGROUND_IMAGE_COMMAND	BACKGROUND_IMAGE
PM_INSERT_BACKGROUND_IMAGE_KEYWORD	element
PM_BACKGROUND_IMAGE_ADD_PRIORITY	500 (integer)
PM_REMOTE_SCHEMATIC_AWS_COMMAND	REMOTE_SCHEMATIC_AWS
PM_REMOTE_ADDITION_AWS_COMMAND	REMOTE_ADDITION_AWS

.

Name i.e. $GLOBALS['<Name>']	Default Value (strings unless otherwise stated)
PM_REMOTE_IMAGE_AWS_SUBCOMMAND	REMOTE_IMAGE_AWS
PM_REMOTE_SCHEMATIC_AWS_SUBCOMMAND	SCHEMATIC
PM_REMOTE_TIMEOUT_MIN	5 (integer)
PM_REMOTE_TIMEOUT_MAX	86400 (integer)
PM_REMOTE_DEFAULT_EXPIRY	300 (integer)
PM_INSERT_EFFECT_COMMAND	EFFECTS
PM_REVEAL_EFFECT_KEYWORD	reveal
PM_EFFECT_PRIORITY	500 (integer)
PM_INSERT_FONT_COMMAND	FONT
PM_FONT_ADD_PRIORITY	505 (integer)
PM_HEADER_CONTENT_COMMAND	HEADER_CONTENT
PM_HEADER_CONTENT_TEXT_SUBCOMMAND	TEXT
PM_HEADER_CONTENT_SCRIPT_SUBCOMMAND	SCRIPT
PM_HEADER_CONTENT_ADD_PRIORITY	510 (integer)
PM_FOOTER_CONTENT_COMMAND	FOOTER_CONTENT
PM_FOOTER_CONTENT_TEXT_SUBCOMMAND	TEXT
PM_FOOTER_CONTENT_SCRIPT_SUBCOMMAND	SCRIPT
PM_FOOTER_CONTENT_ADD_PRIORITY	510 (integer)
PM_INSERT_COMMENTS_COMMAND	COMMENTS
PM_INSERT_PAGINATION_COMMAND	PAGINATION
PM_STYLESHEET_ADD_PRIORITY	506 (integer)
PM_INSERT_COMMENTS_COMMAND	COMMENTS
PM_INSERT_PAGINATION_COMMAND	PAGINATION
PM_INSERT_SEARCHBAR_COMMAND	SEARCHBAR
PM_INSERT_COPYRIGHT_STATEMENT_COMMAND	COPYRIGHT
PM_INSERT_STYLING_COMMAND	STYLE
PM_INSERT_WIDGET_COMMAND	WIDGETS
PM_NO_OF_WIDGETS	5 (integer)

.

Appendix A: Command List & Constants

Full command list

The full command list for PoliMorf is listed below. These commands are listed in the file called 'pcom_globals.php' under *theme/theme_dev/main/functions/pcom/*. You can define these in customs functions.

This section also contains other definitions that are contained in 'pcom_defaults.php' where constants are set. Before changing these values, please make a copy of the original file.

Identifiers

Open "<' and close ">" tags are used to represent a placeholder for a text string e.g. <command> would be mean that there is a command string like "NAV" resulting in "%%NAV::".

Type	Format
Schematic Command (default)	%%<command>::
Sub-Command	<command>=[...<syntax>...]:
Keyword	<keyword>={...<syntax>...}:
Attribute	<attribute>="<value>"
Section Name	{{<name> }}
Schematic Command (called in Section)	[%<command>:]
Insert	[[<Insert name>]]

Command List

The following command descriptions cover all levels in which they can be called.

Schematic Tags

These are only called at the schematic level.

Name	Value	Description	Required
Header Schematic Tag	///HEADER:	Used to mark the start of the Header Schematic	Yes
Before Main Schematic Tag	///BEFORE:	Used to mark the start of the Before Schematic	Optional
Main Schematic Tag	///MAIN:	Used to mark the start of the Header Schematic	Yes
Sidebar Schematic Tag	///SIDEBAR:	Used to mark the start of the Sidebar Schematic	Optional
After Main Schematic Tag	///AFTER:	Used to mark the start of the After Schematic	Optional
Footer Schematic Tag	///FOOTER:	Used to mark the start of the Footer Schematic	Yes
Remote Schematic Tag	///REMOTE:	Used to mark the start of a remote schematic reference	Optional

NAV

The NAV command only called at the schematic command level:

%%NAV:: (or [%NAV:] for Sections)

SubCommands:

None

Keywords:

See table.

NAV			
Keyword	**Description**	**Attribute**	**Notes**
wpmenu	Display a WordPress menu	None	Will return default menu in WordPress if defined in Admin section
		name	Set to name of menu in WordPress Admin. e.g. if menu is called New Menu, attribute is - wpmenu={ name="New Menu" }:
		class	Optional - you can add a class to the menu
mlk	Display a custom menu link	href	Mandatory
		name	Mandatory - is used as text link
		src	Optional - will replace name with an image sourced at 'src'. Name becomes the alt of the image
		class	Optional - you can add a class to individual links
		fa	Optional - adds a Font Awesome icon. Note: will add to src image if present
searchbar	Display a search bar icon (leading to expandable bar when clicked) on the right hand side of the nav menu	None	As description
logo	Displays logo image in nav area	url	Link to local logo image
		filename	Filename of remote logo image
		region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*
		timeout	Remote link expiry time - limited between 5 and 86400 seconds

.

MENU

The MENU command can be called at the schematic command level:

%%MENU:: (or [%MENU:] for Sections)

or at the sub-command level:

MENU=[ ]:

SubCommands:

None

Keywords:

See table.

MENU			
Keyword	**Description**	**Attribute**	**Use**
wpmenu	Display a WordPress menu	None	Will return default menu in WordPress if defined in Admin section
		name	Set to name of menu in WordPress Admin. e.g. if menu is called New Menu, attribute is - wpmenu={ name="New Menu" }:
		class	Optional - you can add a class to the menu
mlk	Display a custom menu link	href	Mandatory
		name	Mandatory - is used as text link
		src	Optional - will replace name with an image sourced at 'src'. Name becomes the alt of the image
		class	Optional - you can add a class to individual links
		fa	Optional - adds a Font Awesome icon. Note: will add to src image if present
title	Displays title for menu	None	Plain text name - no attribute title={ New Title }:

.

TEXT

The TEXT command can be called at the sub command level:

TEXT=[ ]:

SubCommands:

None

Keywords:

A combination of keywords and HTML [ ] tags as per tables below

TEXT			
Keyword	**Description**	**Attribute**	**Use**
img	Adds img html tag	As per ATTRIBUTE table below	Used in a similar way as adding html tags to raw html
img_link	Adds link html tag with img as the anchor	As per ATTRIBUTE table below	Used in a similar way as adding html tags to raw html
link	Adds link to ref	As per ATTRIBUTE table below	Used in a similar way as adding html tags to raw html

ATTRIBUTES				
General Attributes	**Image Attributes**	**Link Attributes**	**Remote Attributes - img and link**	**Remote Attributes - img_link**
accesskey	fa (will use a Font Awesome icon - value is the class attribute)	text (will be used as anchor text in link)	filename	image_filename
class	alt	charset	region	image_region
contenteditable	crossorigin	coords	bucketname	image_bucketname
contextmenu	height	download	timeout	image_timeout
dir	ismap	href		link_filename
draggable	longdesc	hreflang		link_region
dropzone	sizes	media		link_bucketname
hidden	src	name		link_timeout
id	srcset	rel		

.

ATTRIBUTES				
General Attributes	**Image Attributes**	**Link Attributes**	**Remote Attributes - img and link**	**Remote Attributes - img_link**
lang	usemap	rev		
spellcheck	width	shape		
style		target		
tablindex		type		
title				
translate				

For HTML [ ] tags:

HTML [ ] tags are defined in a global, $GLOBALS[‘PM_POST_HTML_CONVERSIONS’]’.

The contents can be expanded by modifying this array. Note: the last entries deal with how to display commands in text without being processed and how to put html tags into TEXT if you want to use third-party software like Font Awesome. To add more or change this, define the global explicitly with the following array **in the custom functions file**.

```
$GLOBALS[‘PM_POST_HTML_CONVERSIONS’] = array(
  '[a_open]' => '<a ',
  '[a_close]' => '</a>',
  '[span_open]' => '<span ',
  '[span_close]' => '</span>',
  '[p]' => '<p>',
  '[/p]' => '</p>',
  '[i]' => '<em>',
  '[/i]' => '</em>',
  '[b]' => '<strong>',
  '[/b]' => '</strong>',
  '[strike]' => '<span style="text-decoration: line-through;">',
  '[/strike]' => '</span>',
  '[u]' => '<span style="text-decoration: underline;">',
  '[/u]' => '</span>',
  '[h1]' => '<h1>',
```

```
  '[/h1]' => '</h1>',
  '[h2]' => '<h2>',
  '[/h2]' => '</h2>',
  '[h3]' => '<h3>',
  '[/h3]' => '</h3>',
  '[h4]' => '<h4>',
  '[/h4]' => '</h4>',
  '[h5]' => '<h5>',
  '[/h5]' => '</h5>',
  '[ol]' => '<ol>',
  '[/ol]' => '</ol>',
  '[ul]' => '<ul>',
  '[/ul]' => '</ul>',
  '[li]' => '<li>',
  '[/li]' => '</li>',
  '[\n]' => '<br/>',
  '[\2n]' => '<br/><br/>',
  '[\3n]' => '<br/><br/><br/>',
  '[quote]' => '<blockquote>',
  '[/quote]' => '</blockquote>',
  '[code]' => '<code>',
  '[/code]' => '</code>',
  '[pre]' => '<pre>',
  '[/pre]' => '</pre>',
  '[samp]' => '<samp>',
  '[/samp]' => '</samp>',
  '[style]' => '<style><!-- ',
  '[/style]' => ' --></style>',
  '[caption]' => '<div class="wp-caption-pm alignnone"><p class="wp-
caption-text">',
  '[/caption]' => '</p></div>',
  '[sch_cmd]' => '%%',
  '[/sch_cmd]' => '::',
  '[sub_cmd]' => '=[',
  '[/sub_cmd]' => ']:',
  '[keyword]' => '={',
```

```
  '[/keyword]' => '}:',
  '[tag]' => '<',
  '[/tag]' => '>',
  '[fs]' => '/',
  '[fa]' => '<i ',
  '[/fa]' => '></i>'
  '[pm-para]' => '[p]',
  '[/pm-para]' => '[/p]',
  '[UP-LOADS]' => 'UPLOADS',
  '[SITE-HOME]' => 'SITE_HOME',
  '[pm-tag]' => '[',
  '[/pm-tag]' => ']',
  '[script_open]' => '<script ',
  '[/script]' => '</script>,
  '[html_comment]' => '<!--',
  '[/html_comment]' => '-->'
);
```

CONTENT_DEFAULT

The CONTENT_DEFAULT command is called at the schematic command level

%%CONTENT_DEFAULT:: (or [%CONTENT_DEFAULT:] for Sections)

SubCommands:

None

Keywords:

None

CONTENT_META

The CONTENT_META command is called at the schematic command level

%%CONTENT_META:: (or [%CONTENT_META:] for Sections)

SubCommands:

None

Keywords:

Either empty syntax or selected (as per table)

CONTENT_META			
Keyword	**Description**	**Attribute**	**Use**
title	Show title	None	Specifically shows title of post or page
author	Show author	None	Specifically shows post or page author
date	Show date	None	Specifically shows date of post or page
cat	Show category list	None	Specifically shows list of post categories (does not apply to pages)
tag	Show tag list	None	Specifically shows list of post tags (does not apply to pages)

CONTENT_INSERT

The CONTENT_INSERT command is called at the schematic command level

%%CONTENT_INSERT:: (or [%CONTENT_INSERT:] for Sections)

SubCommands:

[[<Insert Name>]] followed by TEXT=[ ]: e.g.

[[Insert-1]]

TEXT=[....]:

[[Insert Before Sign Up]] TEXT=[...]:

Note: The insert name must be used in the content exactly as specified. Any spaces before or after the name will mean it will not be recognised in the content.

Keywords:

None

QUOTE

The QUOTE command can be called at the schematic command level:

%%QUOTE:: (or [%QUOTE:] for Sections)

or at the sub-command level:

QUOTE=[ ]:

SubCommands:

None

Keywords:

See table.

QUOTE			
Keyword	**Description**	**Attribute**	**Use**
body	Main quote text	None	Arguments is HTML [] format text - no links or images processed e.g. [p]This is a [i]quote[/i].[/p]
ref	Text for quote reference or attribution	None	Arguments is HTML [] format text - no links or images processed e.g. [p]This is a [i]quote[/i].[/p]
link	Adds link to ref	As per table below (no remote calls processed)	Create link as you would write HTML but without needing text between tags.

QUOTE, Link Keyword	
General Attributes	**Link Attributes**
accesskey	charset
class	coords
contenteditable	download
contextmenu	href
dir	hreflang

.

QUOTE, Link Keyword	
General Attributes	**Link Attributes**
draggable	media
dropzone	name
hidden	rel
id	rev
lang	shape
spellcheck	target
style	type
tablindex	
title	
translate	

.

SECTION

The SECTION command can be called at the schematic command level:

%%SECTION::

It requires definition of a section anywhere in the syntax to work, i.e:

{{ section-id }}.

Note:

Spaces can be used before and after the section id but the id itself needs to use separators conforming to css rules. Custom classes can be used and a default class of "pm-section" is applied.

Calling a SECTION command within a SECTION command will result in erroneous results.

SubCommands:

Schematic commands defined with identifiers: "[%" and ":]"

Keywords:

None.

BACKGROUND_IMAGE

The BACKGROUND_IMAGE command is called at the schematic command level and only works when placed in the Header Format.

%%BACKGROUND_IMAGE::

SubCommands:

None

Keywords:

See table

BACKGROUND			
Keyword	**Description**	**Attribute**	**Notes**
element	Background image reference	url	Local image reference (or accessible image reference)
		filename	Filename of remote background image
		region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*
		timeout	Remote link expiry time - limited between 5 and 86400 seconds

COPYRIGHT

The COPYRIGHT command is called at the schematic command level

%%COPYRIGHT:: (or [%COPYRIGHT:] for Sections)

SubCommands:

None

Keywords:

None

N-BOX (2, 3 and 4)

The N-BOX command is called at the schematic command level:

%%2-BOX:: (or [%2-BOX:] for Sections)

%%3-BOX:: (or [%3-BOX:] for Sections)

%%4-BOX:: (or [%4-BOX:] for Sections)

And has the following syntax:

%%2-BOX:: BOX1: *..commands* BOX2: *..commands*	%%3-BOX:: BOX1: *..commands* BOX2: *..commands* BOX3: *..commands*	%%4-BOX:: BOX1: *..commands* BOX2: *..commands* BOX3: *..commands* BOX4: *..commands*

SubCommands:

MENU, QUOTE, TEXT, WIDGETS, SEARCHBAR, TITLE

Keywords:

None

STYLE

The STYLE command is called at the schematic command level.

%%STYLE:: (or [%STYLE:] for Sections)

SubCommands:

None

Keywords:

None

STYLESHEET

The STYLESHEET command is called at the schematic command level and only works when placed in the Header Format.

%%STYLESHEET::

SubCommands:

None

Keywords:

See table.

STYLESHEET			
Keyword	**Description**	**Attribute**	**Notes**
ref	Reference to external stylesheet i.e. css file	url	stylesheet href
		filename	Remote file
		region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*
		timeout	Remote link expiry time - limited between 5 and 86400 seconds

Number of Widgets

This can be changed by modifying constant PCOM_NO_OF_WIDGETS in class *pcom_commands* in the file pcom_defaults.php. The default is 5.

REMOTE_SCHEMATIC_AWS

The REMOTE_SCHEMATIC_AWS command is called at the schematic command level and at the sub_command level when used for remote Excerpt Data.

%%REMOTE_SCHEMATIC_AWS:: (or [%REMOTE_SCHEMATIC_AWS:] for Sections)

REMOTE_SCHEMATIC_AWS=[ ]:

SubCommands:

When called at schematic level, SCHEMATIC is the subcommand.

i.e.

%%REMOTE_SCHEMATIC_AWS:: SCHEMATIC=[ ]:

When called in the Excerpt Data field, only the subcommand is used:

REMOTE_SCHEMATIC_AWS=[ ]:

Keywords:

None

SCHEMATIC (or REMOTE_SCHEMATIC_AWS for Excerpt)			
Keyword	**Description**	**Attribute**	**Notes**
None	Syntax directly processed within [ ]:	filename	Remote file
		region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*

REMOTE_IMAGE_AWS

The REMOTE_IMAGE_AWS command is called at the subcommand level.

REMOTE_IMAGE_AWS=[ ]:

SubCommands:

None

Keywords:

None

REMOTE_IMAGE_AWS			
Keyword	**Description**	**Attribute**	**Notes**
None	Syntax directly processed within [ ]:	filename	Remote file
		region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*
		timeout	Remote link expiry time - limited between 5 and 86400 seconds

EFFECTS

The EFFECTS command is called at the schematic command level and only works when placed in the Header Format.

%%EFFECTS::

SubCommands:

None

Keywords:

See table.

EFFECTS			
Keyword	**Description**	**Attribute**	**Notes**
reveal	Reveal effect	class	Class to animate
		settings	Optional - Settings according to library attributes.

FONT

The FONT command is called at the schematic command level and only works when placed in the Header Format.

%%FONT::

SubCommands:

None

Keywords:

See table.

Keyword	**Description**	**Attribute**	**Notes**
None	Syntax directly processed within [ ]:	None	Reference to external font

HEADER_CONTENT

The HEADER_CONTENT command is called at the schematic command level and only works when placed in the Header Format.

%%HEADER_CONTENT::

SubCommands:

TEXT, SRC - see table.

Keywords:

None

HEADER_CONTENT			
Subcommand	**Description**	**Attribute**	**Notes**
TEXT	Text of content. Uses text substitutions similar to CONTENT	Multiple	See $GLOBALS['PM_POST_HTML_CONVERSIONS']'.
SRC		filename	Remote file
	Syntax directly processed within [ ]:	region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*
		timeout	Remote link expiry time - limited between 5 and 86400 seconds

FOOTER_CONTENT

The FOOTER_CONTENT command is called at the schematic command level and only works when placed in the Header Format. Information is placed in the footer.

%%FOOTER_CONTENT::

SubCommands:

TEXT, SRC - see table.

Keywords:

None

FOOTER_CONTENT			
Subcommand	**Description**	**Attribute**	**Notes**
TEXT	Text of content. Uses text substitutions similar to CONTENT	Multiple	See $GLOBALS['PM_POST_HTML_CONVERSIONS']'.
SRC		filename	Remote file
	Syntax directly processed within [ ]:	region	Region of bucket
		bucketname	Bucket containing file which has public/private key access defined in *PoliMorf Settings->Cloud*
		timeout	Remote link expiry time - limited between 5 and 86400 seconds