

HTML TEMPLATES



Clarify

Table of Contents

Overview 3

 What is an HTML template? 4

 Installing a custom HTML template 8

 The helpers.php file 10

 Possible values for text run arrays 11

 A text run array 14

Customizing Templates..... 27

 The @article file 28

 Sample templates for download..... 31

Overview

What is an HTML template?

The Clarify HTML template system provides a way to export content into a variety of custom formats. The primary use for HTML templates is to create HTML content, but templates can be customized to create any type of content that is text-based. For example, you might create a custom template to export to XML, Markdown, MediaWiki or other formats.

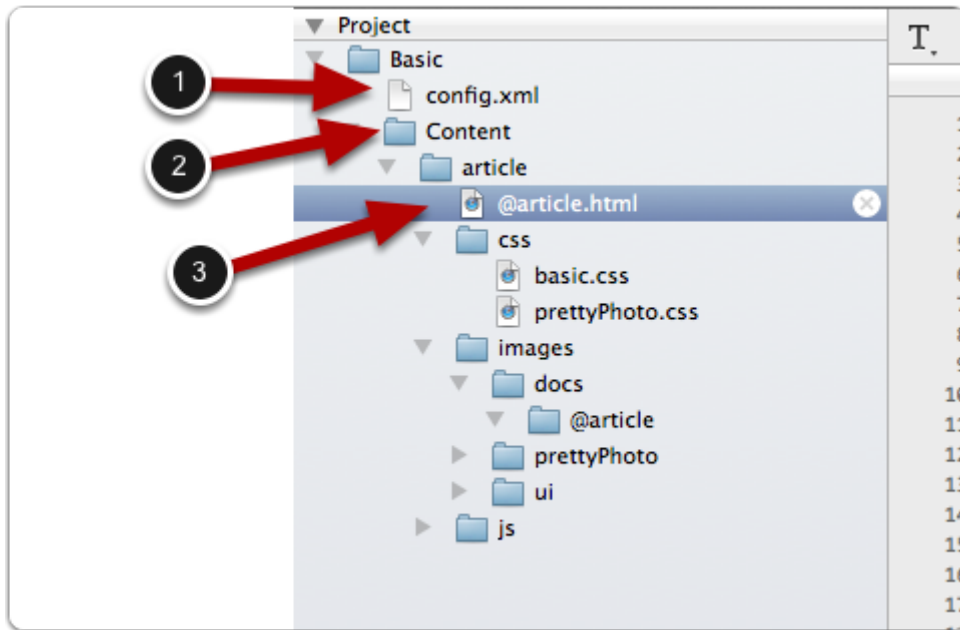
What language does an HTML template use?

Clarify uses the PHP engine to process the template files. Clarify converts manuals and articles into PHP objects that can be accessed from within template scripts.

What does an HTML template folder look like?

An HTML template is a folder which contains the necessary template files. A template folder is comprised of the following items:

1. **config.xml**: The configuration file provides instructions about how Clarify should prepare your content prior to passing it off to the PHP engine.
2. **./Content** folder: The Content folder contains all of the files necessary to export a template. At a minimum it will contain an article folder. If you have supporting PHP files that are used in your HTML template then you would place them in the Content folder.
3. **./Content/article** folder: The article folder contains all of the files that will be exported when exporting an article.



config.xml

The config.xml file can set any of the following properties:

- **article_structure:** controls the nesting of steps in the PHP object. Default value is *hierarchal* which nests substeps under steps. Set to *flat* to get a non-hierarchal list of steps with no nesting.
- **hi_res_images:** specifies whether or not to export images taken on high-resolution monitors with full resolution. The default behavior is *true* in which case all image data is exported but the image dimensions in the PHP object are appropriate for the size that the image should be displayed at. For example, a 600x600 retina image should be displayed at 300x300. Set in *false* to discard the high-resolution image data.
- **image_names:** format used to name images. *random* or *step_title*. *random* can be useful if you need to ensure that you never end up with duplicate image names when importing into a 3rd party system.
- **max_image_dimensions:** The maximum image dimensions to use for width and height. Entries are a comma delimited list of integers. You can provide just the width, width and height, or just height. *Examples:*
600,
600,500
,500
- **text_format:** *xhtml* or *runs*. *xhtml* is appropriate for creating HTML content and you won't need to do anything to the text that Clarify puts into the PHP objects. *runs* formats text in an array that separates the actual text from the formatting applied to the text. This allows you to more easily massage the text into other formats (e.g. markdown).
- **web_safe:** *true* or *false*.
- **word_separator:** character used to separate words in names.

The default config.xml file contains the following XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<properties version="1">
    <web_safe>true</web_safe>
    <text_format>xhtml</text_format>
    <word_separator>-</word_separator>
    <article_structure>hierarchal</article_structure>
    <hi_res_images>true</hi_res_images>
</properties>
```

Here is an example using image_names and max_image_dimensions.

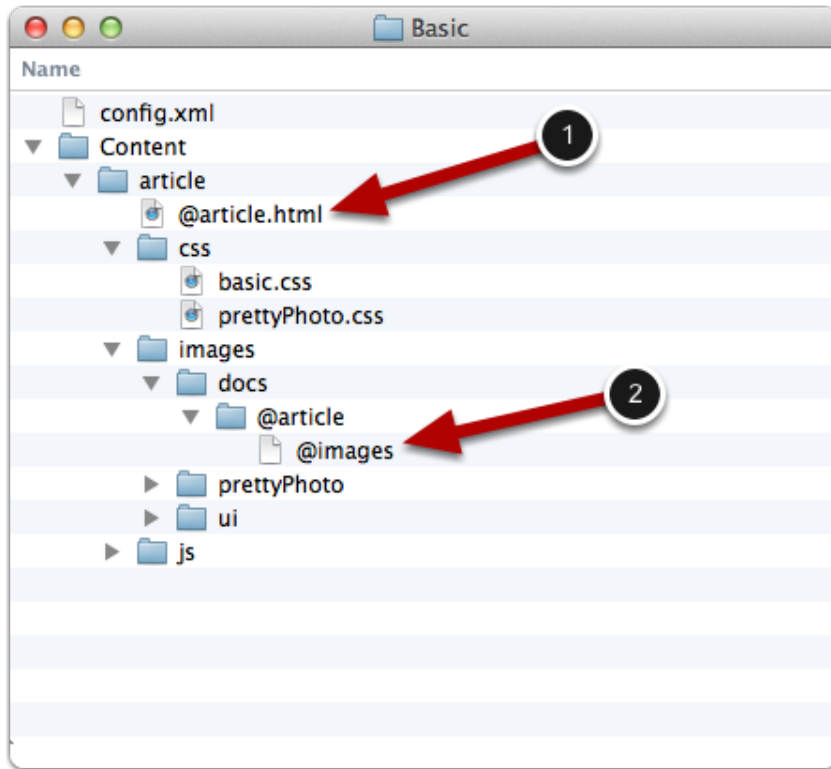
```
<?xml version="1.0" encoding="utf-8" ?>
<properties version="1">
    <web_safe>true</web_safe>
    <text_format>runs</text_format>
    <word_separator>-</word_separator>
    <article_structure>hierarchal</article_structure>
    <hi_res_images>true</hi_res_images>
    <image_names>random</image_names>
    <max_image_dimensions>600,</max_image_dimensions>
</properties>
```

The article folder

The article folder is where you put all of the files that will be exported when exporting an article. There are two files that are required:

1. **@article**: This is the template file that ScreenSteps will process with PHP. The filename must start with @article. The extension you add is up to you.
2. **@images**: This is a placeholder file. ScreenSteps will store all article images in the same location as this file. Notice that in this example the file is in an @article folder. The HTML exporter will replace @article with the name of the article when exporting.

Any other files and folders in the **article** folder will be copied into the folder the article is being exported to.

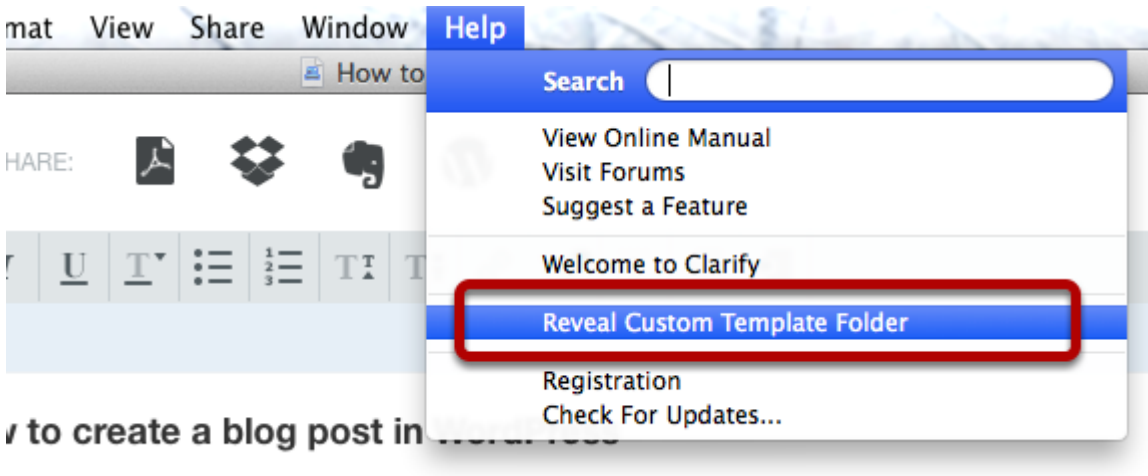


Installing a custom HTML template

To make a custom template available to Clarify, you place the HTML template folder in the Clarify application support folder.

Note that you will need to restart Clarify in order for the template to be available.

Accessing the Template folder



To quickly access the Templates folder use the Help menu and select **Reveal Custom Template Folder** option.

Where HTML templates can go

You can add HTML templates to the following folders:

- HTML: Templates used when exporting to HTML.
- Rich Text Clipboard: Template used when copying a Clarify to the clipboard as rich text. If you put a template folder in here it will override the default template that Clarify uses.
- Web/Dropbox: Templates used for exporting to Dropbox.
- Web/WordPress: Templates used for exporting to WordPress.
- Web Clipboard: Templates that are used with the Clipboard Template setting for Clarify-it.com and Dropbox sharing accounts.



Name	Date Modified	Size	Kind
▶ Application	Oct 15, 2013, 3:44 PM	--	Folder
▶ DOCX	Sep 3, 2013, 2:27 PM	--	Folder
▼ HTML	Today, 11:23 AM	--	Folder
▶ PDF	Jul 22, 2014, 3:23 PM	--	Folder
▶ Rich Text Clipboard	Jan 30, 2014, 5:33 PM	--	Folder
▼ Web	Jan 21, 2014, 1:28 PM	--	Folder
▶ Dropbox	Oct 9, 2013, 9:20 AM	--	Folder
▶ Wordpress	Oct 9, 2013, 9:20 AM	--	Folder
▼ Web Clipboard	Jan 30, 2014, 1:11 PM	--	Folder

The helpers.php file

Every template file that is processed by PHP will have access to the **helpers.php** file that is included with ScreenSteps. This file has a couple of helper functions that will print out content for you. For example, it has a `printArticleXHTML()` function that prints the article as XHTML. It also has a function named `printManualTofCXHTML()` that prints the table of contents as XHTML as well as `printArticleMarkdown()` which outputs a Markdown representation of an article.

Locating the helpers.php file

If you would like to inspect the `helpers.php` file then you will find it inside the ScreenSteps application folder on Windows or application bundle on OS X.

On Windows look in the ScreenSteps installation folder. You will find **helpers.php** in **./components/html_exporter/helpers.php**.

On OS X, it will be located in the application bundle in the **./Contents/Resources/_MacOS/components/html_exporter/helpers.php**.

Possible values for text run arrays

The article lays out the structure and possible values for a text run array. Use this article as a reference when writing PHP code that iterates over a text run array.

A text run array contains any number of paragraphs objects which are referred to by *PARAGRAPH_KEY* below.

[PARAGRAPH_KEY]->**metadata**->**style**: empty or 'code'.

[PARAGRAPH_KEY]->**style**->**align**: empty, 'left', 'right', 'center'. Always empty right now.

[PARAGRAPH_KEY]->**style**->**list_style**: = empty, 'disc' or 'decimal'.

[PARAGRAPH_KEY]->**style**->**list_depth**: = Integer specifying the indentation for the list.

[PARAGRAPH_KEY]->**style**->**list_index**: = Integer specifying the starting number for the list.

[PARAGRAPH_KEY]->**style**->**left_indent**: = A positive value if the text is indented.

[PARAGRAPH_KEY]->**runs**: An array of run objects. Referred to as *[RUN_KEY]* below.

[RUN_KEY]->**style**->**font_family**: The font to use for the run of text.

[RUN_KEY]->**style**->**font_size**: The size of the font.

[RUN_KEY]->**style**->**font_styles**: empty or any combination of 'italic', 'bold', and 'underline' (e.g. 'bold,italic');

[RUN_KEY]->**style**->**color**: The color of the text in RGB format.

[RUN_KEY]->**style**->**text_shift**: empty or an integer <> 0. If the integer is positive then the text is subscript. If the integer is negative then the text is superscript.

[RUN_KEY]->**style**->**link**: The hyperlink assigned to the text.

[RUN_KEY]->**metadata**->**style**: empty or 'code'

[RUN_KEY]->**text**: The text.

Example PHP code for iterating over a text run array

Here is some sample code that iterates over a text run array and generates markup appropriate for MediaWiki.

```
function printTextRunAsMediaWiki($textrun, $type='instructions') {
    $output = '';
    $listDepth = 0;

    // If there is no text then just return an empty string.
    if (!is_array($textrun)) return '';

    // Iterate through paragraphs.
    foreach($textrun as $para)
    {
```

```
$closingPara = '';

/* Unused
$para->style->align
*/

// Is this paragraph formatted as code?
// If not is it a list item?
if ($para->metadata->style == 'code')
{
    $output .= '<code>';
    $closingPara = '</code>';
} else {
    switch ($para->style->list_style)
    {
        case 'decimal':
            $output .= str_repeat('#', $para->style->list_depth) . ' ';
            break;
        default:
            $output .= str_repeat('*', $para->style->list_depth) . ' ';
            break;
    }
}

// Iterate through each text run in the paragraph.
if (isset($para->runs))
{
    foreach ($para->runs as $run)
    {
        $closingRun = '';
        $prefix = '';
        $suffix = '';
        $styles = explode(',', $run->style->font_styles);

        $hasBold = array_search('bold', $styles) != FALSE;
        $hasItalic = array_search('italic', $styles) != FALSE;
        $hasUnderline = array_search('underline', $styles) != FALSE;

        if (!empty($run->style->color))
        {
            $output .= '<span style="color: rgb(' . $run->style->color . ');">';
            $closingRun = '</span>';
        }

        if ($hasItalic) { $prefix .= "''"; $suffix = "''" . $suffix; }
        if ($hasBold) { $prefix .= "''"; $suffix = "''" . $suffix; }
```

```
        if ($hasUnderline) { $prefix .= '<ul>'; $suffix = '</ul>' . $suffix; }

        /* Unused
        $run->style->font_family
        $run->style->font_size
        $run->style->text_shift
        */

        $output .= $prefix;
        $output .= $run->text;
        $output .= $suffix;
        $output .= $closingRun;
    }
}

$output .= $closingPara;

if ($type != 'title') $output .= PHP_EOL;
}

return $output;
}
```

A text run array

If an HTML template has the config.xml **text_format** parameter set to **runs** then text will be formatted using an array. This article provides a brief introduction to how the array is structured and what some examples look like.

If you just plan on using **xhtml** as the setting for **text_format** then you don't need to worry about the information in this article.

Below you will see some basic example text followed by the **print_r** output for the array in PHP. Each paragraph in the text becomes a key in the array. The first paragraph starts at [0].

A paragraph can have properties. For example, meta->style can be *code* which means the paragraph should be formatted as code. Or the style->list_style might be set to *disc* in which case it is a bulleted list.

Each paragraph is then broken up into **runs**. A **run** is a run of text that shares the same properties, such as color and style. Whenever a property of the text changes a new entry is added to the **runs** array for the paragraph.

By organizing the text this way you can iterate through a text run array in a straightforward way and format the according to the specification of the format you are exporting to.

Example text 1

This is some *example* text.

This is another paragraph in the example text.

Text run array 1

```
Array
(
    [0] => stdClass Object
        (
            [metadata] => stdClass Object
                (
                    [style] =>
                )
            [runs] => Array
                (
```

```
[0] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
        [link] =>
    )
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [text] => This is some
)
[1] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] => bold,italic
        [text_shift] =>
        [link] =>
    )
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [text] => example
)
[2] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
        [link] =>
    )
    [metadata] => stdClass Object
```

```
(
    [style] =>
)
[text] => text.
)
)
[style] => stdClass Object
(
    [align] =>
    [list_depth] => 0
    [list_style] =>
    [list_index] => 0
    [left_indent] => 0
)
)
[1] => stdClass Object
(
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [runs] => Array
    (
        [0] => stdClass Object
        (
            [style] => stdClass Object
            (
                [color] =>
                [font_family] =>
                [font_size] =>
                [font_styles] =>
                [text_shift] =>
                [link] =>
            )
            [metadata] => stdClass Object
            (
                [style] =>
            )
            [text] => This is another paragraph in the example text.
        )
    )
    [style] => stdClass Object
    (
        [align] =>
        [list_depth] => 0
        [list_style] =>
```



```
[list_index] => 0
[left_indent] => 0
    )
)
)
```

More extensive example

Below is a more extensive example that uses lists and code.

Example text 2

This is *some **introductory*** text.

- List item
- List item

The text **continues** on.

1. List item
 1. List item
 2. List item
2. List item

Some more text.

3. List item with hard coded start number.

```
This is some code.
```

Text run array 2

```
Array
(
    [0] => stdClass Object
        (
            [metadata] => stdClass Object
                (
                    [style] =>
                )
            [runs] => Array
                (
```

```
[0] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
    )
    [text] => This is
)
[1] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] => italic
        [text_shift] =>
    )
    [text] => some
)
[2] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
    )
    [text] =>
)
[3] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] => bold,italic
        [text_shift] =>
    )
)
```

```
        [text] => introductory
    )
[4] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
    )
    [text] => text.
)
)
[style] => stdClass Object
(
    [align] =>
    [list_depth] => 0
    [list_style] =>
    [list_index] => 0
)
)
[1] => stdClass Object
(
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [runs] => Array
    (
        [0] => stdClass Object
        (
            [style] => stdClass Object
            (
                [color] =>
                [font_family] =>
                [font_size] =>
                [font_styles] =>
                [text_shift] =>
            )
            [text] => List item
        )
    )
    [style] => stdClass Object
    (
```

```
        [align] =>
        [list_depth] => 1
        [list_style] => disc
        [list_index] => 0
    )
)
[2] => stdClass Object
(
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [runs] => Array
    (
        [0] => stdClass Object
        (
            [style] => stdClass Object
            (
                [color] =>
                [font_family] =>
                [font_size] =>
                [font_styles] =>
                [text_shift] =>
            )
            [text] => List item
        )
    )
    [style] => stdClass Object
    (
        [align] =>
        [list_depth] => 1
        [list_style] => disc
        [list_index] => 0
    )
)
[3] => stdClass Object
(
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [runs] => Array
    (
        [0] => stdClass Object
        (
            [style] => stdClass Object
```

```
(
    [color] =>
    [font_family] =>
    [font_size] =>
    [font_styles] =>
    [text_shift] =>
)
[text] => The text
)
[1] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] => 255,7,44
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
    )
    [text] => continues
)
[2] => stdClass Object
(
    [style] => stdClass Object
    (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
    )
    [text] => on.
)
)
[style] => stdClass Object
(
    [align] =>
    [list_depth] => 0
    [list_style] =>
    [list_index] => 0
)
)
[4] => stdClass Object
(
    [metadata] => stdClass Object
    (
```

```
        [style] =>
      )
    [runs] => Array
      (
        [0] => stdClass Object
          (
            [style] => stdClass Object
              (
                [color] =>
                [font_family] =>
                [font_size] =>
                [font_styles] =>
                [text_shift] =>
              )
            [text] => List item
          )
        )
      [style] => stdClass Object
        (
          [align] =>
          [list_depth] => 1
          [list_style] => decimal
          [list_index] => 0
        )
      )
    [5] => stdClass Object
      (
        [metadata] => stdClass Object
          (
            [style] =>
          )
        [runs] => Array
          (
            [0] => stdClass Object
              (
                [style] => stdClass Object
                  (
                    [color] =>
                    [font_family] =>
                    [font_size] =>
                    [font_styles] =>
                    [text_shift] =>
                  )
                [text] => List item
              )
            )
          )
        )
      )
```

```
[style] => stdClass Object
(
    [align] =>
    [list_depth] => 2
    [list_style] => decimal
    [list_index] => 0
)
)
[6] => stdClass Object
(
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [runs] => Array
    (
        [0] => stdClass Object
        (
            [style] => stdClass Object
            (
                [color] =>
                [font_family] =>
                [font_size] =>
                [font_styles] =>
                [text_shift] =>
            )
            [text] => List item
        )
    )
    [style] => stdClass Object
    (
        [align] =>
        [list_depth] => 2
        [list_style] => decimal
        [list_index] => 0
    )
)
[7] => stdClass Object
(
    [metadata] => stdClass Object
    (
        [style] =>
    )
    [runs] => Array
    (
        [0] => stdClass Object
```

```
(
  [style] => stdClass Object
  (
    [color] =>
    [font_family] =>
    [font_size] =>
    [font_styles] =>
    [text_shift] =>
  )
  [text] => List item
)
[style] => stdClass Object
(
  [align] =>
  [list_depth] => 1
  [list_style] => decimal
  [list_index] => 0
)
)
[8] => stdClass Object
(
  [metadata] => stdClass Object
  (
    [style] =>
  )
  [runs] => Array
  (
    [0] => stdClass Object
    (
      [style] => stdClass Object
      (
        [color] =>
        [font_family] =>
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
      )
      [text] => Some more text.
    )
  )
  [style] => stdClass Object
  (
    [align] =>
    [list_depth] => 0
    [list_style] =>
```



```
        [list_index] => 0
      )
    )
  [9] => stdClass Object
  (
    [metadata] => stdClass Object
    (
      [style] =>
    )
    [runs] => Array
    (
      [0] => stdClass Object
      (
        [style] => stdClass Object
        (
          [color] =>
          [font_family] =>
          [font_size] =>
          [font_styles] =>
          [text_shift] =>
        )
        [text] => List item with hard coded start number.
      )
    )
    [style] => stdClass Object
    (
      [align] =>
      [list_depth] => 1
      [list_style] => decimal
      [list_index] => 0
    )
  )
[10] => stdClass Object
(
  [metadata] => stdClass Object
  (
    [style] => code
  )
  [runs] => Array
  (
    [0] => stdClass Object
    (
      [style] => stdClass Object
      (
        [color] =>
        [font_family] => Courier
      )
    )
  )
)
```

```
        [font_size] =>
        [font_styles] =>
        [text_shift] =>
    )
    [text] => This is some code.
)
)
[style] => stdClass Object
(
    [align] =>
    [list_depth] => 0
    [list_style] =>
    [list_index] => 0
)
)
)
```

Customizing Templates

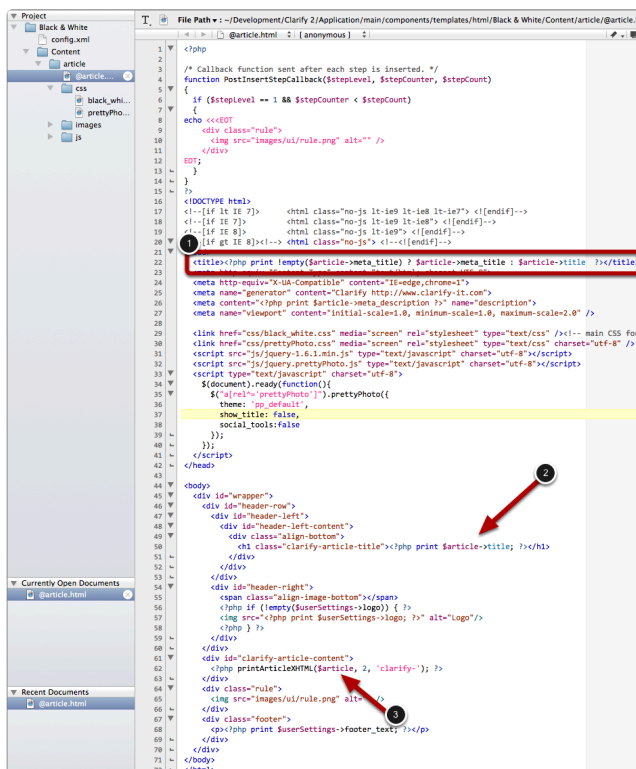
The @article file

The @article template file is processed by PHP and will generate the content for each article that you are exporting.

An example

Here is the @article.html for the Black & White template that ships with Clarify. As you can see, it is a mixture of HTML and PHP. The document content and user settings are all available as PHP objects that can be used to generate the output. For example:

1. PHP is used to print the article title. If a meta title has been set then the meta title is used.
2. The article title is added to the document. This is the title the user sees on the browser page.
3. Article content is printed out using a function from [helpers.php](#).



```

1  <?php
2  /* Callback function sent after each step is inserted. */
3  function PostInsertStepCallback($stepLevel, $stepCounter, $stepCount)
4  {
5      if ($stepLevel == 1 && $stepCounter < $stepCount)
6      {
7          echo <<EOT
8              <div class="rule">
9                  
10             </div>
11         EOT;
12     }
13 }
14 }
15 }
16
17 <!--[[ If IE 7]]> <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif-->
18 <!--[[ If IE 8]]> <html class="no-js lt-ie9 lt-ie8"> <![endif-->
19 <!--[[ If IE 8]]> <html class="no-js lt-ie9"> <![endif-->
20 <!--[[ If IE 8]]> <html class="no-js"> <!--[[endif-->
21
22 <title>?php print empty($article->meta_title) ? $article->meta_title : $article->title ?></title>
23
24 <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
25 <meta name="generator" content="Clarify http://www.clarify-it.com">
26 <meta content="?php print $article->meta_description ?>" name="description">
27 <meta name="viewport" content="initial-scale=1.0, minimum-scale=1.0, maximum-scale=2.0"> />
28
29 <link href="css/black.white.css" media="screen" rel="stylesheet" type="text/css" /> <!-- main CSS for
30 <link href="css/prettyPhoto.css" media="screen" rel="stylesheet" type="text/css" />
31 <script src="js/jquery-1.6.1.min.js" type="text/javascript" charset="utf-8"></script>
32 <script src="js/jquery.prettyPhoto.js" type="text/javascript" charset="utf-8"></script>
33 <script type="text/javascript" charset="utf-8">
34 $(document).ready(function(){
35     if( $('#prettyPhoto') ){ prettyPhoto({
36         theme: 'pp_default',
37         show_title: false,
38         social_tools: false
39     });
40 }
41 </script>
42 </head>
43
44 <body>
45 <div id="wrapper">
46 <div id="header-row">
47 <div id="header-left">
48 <div id="header-left-content">
49 <div class="align-bottom">
50 <h1 class="clarify-article-title">?php print $article->title; ?></h1>
51 </div>
52 </div>
53 </div>
54 <div id="header-right">
55 <span class="align-image-bottom"></span>
56 <php if (empty($userSettings->logo)) { ?>
57 
58 <php ?>
59 </div>
60 </div>
61 <div id="clarify-article-content">
62 <php printArticleHTML($article, 2, 'clarify'); ?>
63 </div>
64 <div class="rule">
65 
66 </div>
67 <div class="footer">
68 <span?php print $userSettings->footer_text; ?></span>
69 </div>
70 </div>
71 </body>
72 </html>

```

Which PHP objects are available to the @article file?

The following variables are available in the file:

- \$article: The article content.

- `$userSettings`: Settings the user has configured.
- `$output_filename`: The name of the file that the template output is being saved to.

`$article object (hierarchical)`

```
stdClass Object
(
    [title] => The title of the article.
    [title_websafe] => The websafe version of the article title. Safe for use in a URL.
    [description] => The article description.
    [description_plain] => The article description with no formatting applied. This
only applies if text_format is set to xhtml. It will not be present when set to runs.
    [id] => An integer.
    [meta_description] => The meta description assigned to the article.
    [meta_search] => The meta search assigned to the article.
    [meta_title] => The meta title assigned to the article.
    [tag_list] => A comma delimited list of tags assigned to the article.
    [tags] => Array
        (
            [0] => Tag name.
        )
    [steps] => Array
        (
            [0] => stdClass Object
                (
                    [id] => An integer.
                    [anchor_name] => The step anchor name.
                    [instructions] => The step instructions.
                    [instructions_plain] => The step instructions with no formatting
applied. This only applies if text_format is set to xhtml. It will be be present when
set to runs.
                    [instructions_position] => 'above' or 'below'.
                    [level] => 1 or 2. 2 means the step is a sub-step of the preceding
step. This is useful if the HTML template article_structure property is set to 'flat'.
                    [media] => stdClass Object
                        (
                            [fullsize] => stdClass Object
                                (
                                    [type] => image'.
                                    [filename] => The full path to the step image.
                                    [relative_filename] => The relative path to the
step image.
                                    [url] => If the template is being used to publish
to a service like WordPress then this is the URL where the image is located.
                                    [width] => The width of the image in pixels.
```

```

        [height] => The height of the image in pixels.
    )
    [thumbnail] =>
    (
        [type] => image'.
        [filename] => The full path to the step thumbnail
image.

        [relative_filename] => The relative path to the
step thumbnail image.

        [url] => If the template is being used to publish
to a service like WordPress then this is the URL where the image is located.
        [height] => The height of the image in pixels.
        [width] => The width of the image in pixels.
    )
    [type] => 'image' or 'html'.
    [url_for_nonhtml] =>
    [html] => If 'type' is 'html' then this contains the HTML
for the step.

    )
    [media_alt] => The alternate tag for the media.
    [title] => The step title.
    [title_websafe] => The websafe version of the step title. Safe for
use in a URL.

    [uuid] => The UUID of the step.
    [substep] => Array: this is only present if the HTML template
article_structure property is set to 'hierarchal' (default).
    (
        [0] => stdClass Object
        (
            same structure as a step...
        )
    )
)

```

\$userSettings object

```

stdClass Object
(
    [footer_text] => The text the use wants to display in the footer.
    [logo] => The path to the logo file the user selected.
)

```

Sample templates for download

This article contains links to sample templates that you can use as a starting point.

Black & White

This is a template that ships with Clarify. It generates HTML files.

<http://files.clarify-it.com/v2/templates/html/Black%20%26%20White.zip>

Markdown Hi-Res

This template shows how to customize your own Markdown template. It includes a `markdown_process.php` file with the template that includes the image width and height parameters in the image reference. In addition, the template `hi_res_images` property is set to `true` so that the full-size image is exported for screen captures taken on high-resolution monitors.

<http://files.clarify-it.com/v2/templates/html/Markdown%20Hi-Res.zip>

Markdown Passthru

This template passes the text in the Clarify document directly through with only minor modifications. The document and step titles have Markdown added but the description and instructions pass through. This allows you to write your Clarify document in Markdown and then use this HTML template to export it as-is.

<http://files.clarify-it.com/v2/templates/html/Markdown%20Passthru.zip>

Shadowbox

This template outputs HTML with images that are tagged for use with Shadowbox. The HTML is appropriate for WordPress export. The template will create a full size version of any image over 540 pixels and if Shadowbox is available then clicking on the thumbnail version will display the full-size image. You can change the maximum image dimensions in the `config.xml` file.

This template uses a copy of the `printArticleHTML` function from the `helpers.php` file so that the output can be customized.

<http://files.clarify-it.com/v2/templates/html/Shadowbox.zip>