**HTML Helper**

The HTML Helper file contains functions that assist in working with HTML.

* [Loading this Helper](https://codeigniter.com/user_guide/helpers/html_helper.html#loading-this-helper)
* [Available Functions](https://codeigniter.com/user_guide/helpers/html_helper.html#available-functions)

[**Loading this Helper**](https://codeigniter.com/user_guide/helpers/html_helper.html#id1)

This helper is loaded using the following code:

$this**->**load**->**helper('html');

[**Available Functions**](https://codeigniter.com/user_guide/helpers/html_helper.html#id2)

The following functions are available:

**heading([*$data = ''*[, *$h = '1'*[, *$attributes = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*string*) – Content * **$h** (*string*) – Heading level * **$attributes** (*mixed*) – HTML attributes |
| **Returns:** | HTML heading tag |
| **Return type:** | string |

Lets you create HTML heading tags. The first parameter will contain the data, the second the size of the heading. Example:

**echo** heading('Welcome!', 3);

The above would produce: <h3>Welcome!</h3>

Additionally, in order to add attributes to the heading tag such as HTML classes, ids or inline styles, a third parameter accepts either a string or an array:

**echo** heading('Welcome!', 3, 'class="pink"');

**echo** heading('How are you?', 4, **array**('id' **=>** 'question', 'class' **=>** 'green'));

The above code produces:

<h3 class**=**"pink">Welcome!<h3>

<h4 id**=**"question" class**=**"green">How are you?</h4>

**img([*$src = ''*[, *$index\_page = FALSE*[, *$attributes = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$src** (*string*) – Image source data * **$index\_page** (*bool*) – Whether to treat $src as a routed URI string * **$attributes** (*array*) – HTML attributes |
| **Returns:** | HTML image tag |
| **Return type:** | string |

Lets you create HTML <img /> tags. The first parameter contains the image source. Example:

**echo** img('images/picture.jpg'); *// gives <img src="http://site.com/images/picture.jpg" />*

There is an optional second parameter that is a TRUE/FALSE value that specifics if the *src* should have the page specified by $config['index\_page'] added to the address it creates. Presumably, this would be if you were using a media controller:

**echo** img('images/picture.jpg', **TRUE**); *// gives <img src="http://site.com/index.php/images/picture.jpg" alt="" />*

Additionally, an associative array can be passed to the img() function for complete control over all attributes and values. If an *alt* attribute is not provided, CodeIgniter will generate an empty string.

Example:

$image\_properties **=** **array**(

'src' **=>** 'images/picture.jpg',

'alt' **=>** 'Me, demonstrating how to eat 4 slices of pizza at one time',

'class' **=>** 'post\_images',

'width' **=>** '200',

'height'**=>** '200',

'title' **=>** 'That was quite a night',

'rel' **=>** 'lightbox'

);

img($image\_properties);

*// <img src="http://site.com/index.php/images/picture.jpg" alt="Me, demonstrating how to eat 4 slices of pizza at one time" class="post\_images" width="200" height="200" title="That was quite a night" rel="lightbox" />*

**link\_tag([*$href = ''*[, *$rel = 'stylesheet'*[, *$type = 'text/css'*[, *$title = ''*[, *$media = ''*[, *$index\_page = FALSE*]]]]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$href** (*string*) – What are we linking to * **$rel** (*string*) – Relation type * **$type** (*string*) – Type of the related document * **$title** (*string*) – Link title * **$media** (*string*) – Media type * **$index\_page** (*bool*) – Whether to treat $src as a routed URI string |
| **Returns:** | HTML link tag |
| **Return type:** | string |

Lets you create HTML <link /> tags. This is useful for stylesheet links, as well as other links. The parameters are *href*, with optional *rel*, *type*, *title*, *media* and *index\_page*.

*index\_page* is a boolean value that specifies if the *href* should have the page specified by $config['index\_page'] added to the address it creates.

Example:

**echo** link\_tag('css/mystyles.css');

*// gives <link href="http://site.com/css/mystyles.css" rel="stylesheet" type="text/css" />*

Further examples:

**echo** link\_tag('favicon.ico', 'shortcut icon', 'image/ico');

*// <link href="http://site.com/favicon.ico" rel="shortcut icon" type="image/ico" />*

**echo** link\_tag('feed', 'alternate', 'application/rss+xml', 'My RSS Feed');

*// <link href="http://site.com/feed" rel="alternate" type="application/rss+xml" title="My RSS Feed" />*

Additionally, an associative array can be passed to the link() function for complete control over all attributes and values:

$link **=** **array**(

'href' **=>** 'css/printer.css',

'rel' **=>** 'stylesheet',

'type' **=>** 'text/css',

'media' **=>** 'print'

);

**echo** link\_tag($link);

*// <link href="http://site.com/css/printer.css" rel="stylesheet" type="text/css" media="print" />*

**ul(*$list*[, *$attributes = ''*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$list** (*array*) – List entries * **$attributes** (*array*) – HTML attributes |
| **Returns:** | HTML-formatted unordered list |
| **Return type:** | string |

Permits you to generate unordered HTML lists from simple or multi-dimensional arrays. Example:

$list **=** **array**(

'red',

'blue',

'green',

'yellow'

);

$attributes **=** **array**(

'class' **=>** 'boldlist',

'id' **=>** 'mylist'

);

**echo** ul($list, $attributes);

The above code will produce this:

<ul class**=**"boldlist" id**=**"mylist">

<li>red</li>

<li>blue</li>

<li>green</li>

<li>yellow</li>

</ul>

Here is a more complex example, using a multi-dimensional array:

$attributes **=** **array**(

'class' **=>** 'boldlist',

'id' **=>** 'mylist'

);

$list **=** **array**(

'colors' **=>** **array**(

'red',

'blue',

'green'

),

'shapes' **=>** **array**(

'round',

'square',

'circles' **=>** **array**(

'ellipse',

'oval',

'sphere'

)

),

'moods' **=>** **array**(

'happy',

'upset' **=>** **array**(

'defeated' **=>** **array**(

'dejected',

'disheartened',

'depressed'

),

'annoyed',

'cross',

'angry'

)

)

);

**echo** ul($list, $attributes);

The above code will produce this:

<ul class**=**"boldlist" id**=**"mylist">

<li>colors

<ul>

<li>red</li>

<li>blue</li>

<li>green</li>

</ul>

</li>

<li>shapes

<ul>

<li>round</li>

<li>suare</li>

<li>circles

<ul>

<li>elipse</li>

<li>oval</li>

<li>sphere</li>

</ul>

</li>

</ul>

</li>

<li>moods

<ul>

<li>happy</li>

<li>upset

<ul>

<li>defeated

<ul>

<li>dejected</li>

<li>disheartened</li>

<li>depressed</li>

</ul>

</li>

<li>annoyed</li>

<li>cross</li>

<li>angry</li>

</ul>

</li>

</ul>

</li>

</ul>

**ol(*$list*, *$attributes = ''*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$list** (*array*) – List entries * **$attributes** (*array*) – HTML attributes |
| **Returns:** | HTML-formatted ordered list |
| **Return type:** | string |

Identical to [ul()](https://codeigniter.com/user_guide/helpers/html_helper.html#ul), only it produces the <ol> tag for ordered lists instead of <ul>.

**meta([*$name = ''*[, *$content = ''*[, *$type = 'name'*[, *$newline = "n"*]]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$name** (*string*) – Meta name * **$content** (*string*) – Meta content * **$type** (*string*) – Meta type * **$newline** (*string*) – Newline character |
| **Returns:** | HTML meta tag |
| **Return type:** | string |

Helps you generate meta tags. You can pass strings to the function, or simple arrays, or multidimensional ones.

Examples:

**echo** meta('description', 'My Great site');

*// Generates: <meta name="description" content="My Great Site" />*

**echo** meta('Content-type', 'text/html; charset=utf-8', 'equiv');

*// Note the third parameter. Can be "equiv" or "name"*

*// Generates: <meta http-equiv="Content-type" content="text/html; charset=utf-8" />*

**echo** meta(**array**('name' **=>** 'robots', 'content' **=>** 'no-cache'));

*// Generates: <meta name="robots" content="no-cache" />*

$meta **=** **array**(

**array**(

'name' **=>** 'robots',

'content' **=>** 'no-cache'

),

**array**(

'name' **=>** 'description',

'content' **=>** 'My Great Site'

),

**array**(

'name' **=>** 'keywords',

'content' **=>** 'love, passion, intrigue, deception'

),

**array**(

'name' **=>** 'robots',

'content' **=>** 'no-cache'

),

**array**(

'name' **=>** 'Content-type',

'content' **=>** 'text/html; charset=utf-8', 'type' **=>** 'equiv'

)

);

**echo** meta($meta);

*// Generates:*

*// <meta name="robots" content="no-cache" />*

*// <meta name="description" content="My Great Site" />*

*// <meta name="keywords" content="love, passion, intrigue, deception" />*

*// <meta name="robots" content="no-cache" />*

*// <meta http-equiv="Content-type" content="text/html; charset=utf-8" />*

**doctype([*$type = 'xhtml1-strict'*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$type** (*string*) – Doctype name |
| **Returns:** | HTML DocType tag |
| **Return type:** | string |

Helps you generate document type declarations, or DTD’s. XHTML 1.0 Strict is used by default, but many doctypes are available.

Example:

**echo** doctype(); *// <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">*

**echo** doctype('html4-trans'); *// <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">*

The following is a list of doctype choices. These are configurable, and pulled from application/config/doctypes.php

| **Document type** | **Option** | **Result** |
| --- | --- | --- |
| XHTML 1.1 | xhtml11 | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.1//EN” “[http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd](https://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd)”> |
| XHTML 1.0 Strict | xhtml1-strict | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN” “[http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd](https://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd)”> |
| XHTML 1.0 Transitional | xhtml1-trans | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN” “[http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd](https://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd)”> |
| XHTML 1.0 Frameset | xhtml1-frame | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Frameset//EN” “[http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd](https://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd)”> |
| XHTML Basic 1.1 | xhtml-basic11 | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML Basic 1.1//EN” “[http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd](https://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd)”> |
| HTML 5 | html5 | <!DOCTYPE html> |
| HTML 4 Strict | html4-strict | <!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01//EN” “[http://www.w3.org/TR/html4/strict.dtd](https://www.w3.org/TR/html4/strict.dtd)”> |
| HTML 4 Transitional | html4-trans | <!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01 Transitional//EN” “[http://www.w3.org/TR/html4/loose.dtd](https://www.w3.org/TR/html4/loose.dtd)”> |
| HTML 4 Frameset | html4-frame | <!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01 Frameset//EN” “[http://www.w3.org/TR/html4/frameset.dtd](https://www.w3.org/TR/html4/frameset.dtd)”> |
| MathML 1.01 | mathml1 | <!DOCTYPE math SYSTEM “[http://www.w3.org/Math/DTD/mathml1/mathml.dtd](https://www.w3.org/Math/DTD/mathml1/mathml.dtd)”> |
| MathML 2.0 | mathml2 | <!DOCTYPE math PUBLIC “-//W3C//DTD MathML 2.0//EN” “[http://www.w3.org/Math/DTD/mathml2/mathml2.dtd](https://www.w3.org/Math/DTD/mathml2/mathml2.dtd)”> |
| SVG 1.0 | svg10 | <!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.0//EN” “[http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd](https://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd)”> |
| SVG 1.1 Full | svg11 | <!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1//EN” “[http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd](https://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd)”> |
| SVG 1.1 Basic | svg11-basic | <!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1 Basic//EN” “[http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-basic.dtd](https://www.w3.org/Graphics/SVG/1.1/DTD/svg11-basic.dtd)”> |
| SVG 1.1 Tiny | svg11-tiny | <!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1 Tiny//EN” “[http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-tiny.dtd](https://www.w3.org/Graphics/SVG/1.1/DTD/svg11-tiny.dtd)”> |
| XHTML+MathML+SVG (XHTML host) | xhtml-math-svg-xh | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN” “[http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd](https://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd)”> |
| XHTML+MathML+SVG (SVG host) | xhtml-math-svg-sh | <!DOCTYPE svg:svg PUBLIC “-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN” “http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd”> |
| XHTML+RDFa 1.0 | xhtml-rdfa-1 | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML+RDFa 1.0//EN” “[http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd](https://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd)”> |
| XHTML+RDFa 1.1 | xhtml-rdfa-2 | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML+RDFa 1.1//EN” “[http://www.w3.org/MarkUp/DTD/xhtml-rdfa-2.dtd](https://www.w3.org/MarkUp/DTD/xhtml-rdfa-2.dtd)”> |