

[<< Back to Dashboard](#)

## Exploring the HTMLHelper Plugin

### Overview

A cool utility that helps you when working with HTML, from creating doc types, to managing your js/css assets, to rendering tables and lists from data

### img

Create image tags using the SES base URL or not.

#### Returns

- This function returns *any*

#### Arguments

Key	Type	Required	Default	Description
src	any	Yes	---	The source URL to link to or a structure of name-value pairs to use to construct the image: [src,alt,class,width,height,title,rel]
alt	string	No		The alt tag
class	string	No		The class tag
width	string	No		The width tag
height	string	No		The height tag
title	string	No		The title tag
rel	string	No		The rel tag
noBaseURL	boolean	No	false	Prepends the setting of baseURL if false

#### Examples

The first parameter *src* can be used to pass in the source URL of the image or a structure of name-value pairs to use in the construction of the image.

```
#html.img('includes/images/picture.png')#
// produces 

#html.img(src="includes/images/pic.png",noBaseURL=false)#
// produces 

#html.img('includes/images/picture.png',"image","myImages",200,200,"Cool Image","friend")#
// produces 
```

You can also use the *src* argument as a cool structure of name value pairs that match the arguments in the function:

```
#html.img({img:"includes/images/picture.png",alt:"image",title:"Cool Image"})#
// produces 
```

### slugify

Create a URL safe slug from a string, mostly used to create nice permalinks

#### Returns

- This function returns *string*

#### Arguments

Key	Type	Required	Default	Description
str	string	Yes	---	The string to slugify

#### Examples

You can easily use this tag to generate nice identifier permalink slugs from text.

```
#html.slugify("Sept. Is great- for me--and you")#
//produces sept-is-great-for-me-and-you
```

### tag

Surround content with a tag

#### Returns

- This function returns *any*

#### Arguments

Key	Type	Required	Default	Description
tag	string	Yes	---	The tag to generate
content	string	No		The content of the tag
attributes	struct	No	[runtime expression]	The attributes to use in the tag

#### Examples

You can easily use this tag to generate any HTML/XML tag in a more convenient method:

```
#html.tag("MyTag","MyContent",{class="cool",id="awesome"})#
//produces <MyTag class="cool" id="awesome">MyContent</MyTag>

#html.tag("Property","1000",{name:"maxElements"})#
//produces <Property name="maxElements">1000</Property>
```

### ol

Create ordered lists according to passed in values and arguments, compressed HTML

#### Returns

- This function returns *any*

#### Arguments

Key	Type	Required	Default	Description
values	any	Yes		An array of values or list of values or a query

### Contents

- [Exploring the HTMLHelper Plugin](#)
  - [Overview](#)
  - [img](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [slugify](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [tag](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [ol](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [nl](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [table](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [addStyleContent](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [link](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [hr](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [heading](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [addAsset](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [meta](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [nbs](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [addSContent](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
  - [docType](#)
    - [Returns](#)
    - [Arguments](#)
    - [Examples](#)
    - [More From The Community](#)

attributes	struct	No	[runtime expression]	Attributes for the enclosing OL tags
column	string	No		If the values is a query, this is the name of the column to get the data from to create the list

**Examples**

You can use the *values* element as a simple list of values to represent in a list or actually pass in an array of values or a query:

```
// List
#html.ol(*1,2,3*)#
<ol>
<li>1</li>
<li>2</li>
<li>3</li>
</ol>

//Array
#html.ol([1,2,3])#
<ol>
<li>1</li>
<li>2</li>
<li>3</li>
</ol>

//Query MUST use the column value to tell which value to render
#html.ol(values=myQuery,column=name)#
<ol>
<li>luis</li>
<li>pio</li>
<li>peter</li>
</ol>
```

**ul**

Create un-ordered lists according to passed in values and arguments, compressed HTML

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
values	any	Yes		An array of values or list of values
attributes	struct	No	[runtime expression]	Attributes for the enclosing UL tags
column	string	No		If the values is a query, this is the name of the column to get the data from to create the list

**Examples**

You can use the *values* element as a simple list of values to represent in a list or actually pass in an array of values or a query:

```
// List
#html.ul(*1,2,3*)#
<ul>
<li>1</li>
<li>2</li>
<li>3</li>
</ul>

//Array
#html.ul([1,2,3])#
<ul>
<li>1</li>
<li>2</li>
<li>3</li>
</ul>

//Query MUST use the column value to tell which value to render
#html.ul(values=myQuery,column=name)#
<ul>
<li>luis</li>
<li>pio</li>
<li>peter</li>
</ul>
```

**table**

Convert a table out of data (either a query or array of structures or array of entities)

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
data	any	Yes	---	The query or array of structures or array of entities to convert into a table
includes	string	No		The columns to include in the rendering
excludes	string	No		The columns to exclude in the rendering
attributes	struct	No	[runtime expression]	Attributes for the enclosing table tag

**Examples**

```
// Create a table out of a query with the following columns: [id,name,createDate,modifydate]
#html.table(data=myQuery, includes="name,createDate", attributes={border=1})#

// Create a table out of a query with the following columns: [id,name,createDate,modifydate]
#html.table(data=myQuery, excludes="id", attributes={border=1})#

// Create a table out of an array of structures
data = [
  {id=1,name="luis"},
  {id=2,name="luis2"}
]
#html.table(data=data, attributes={border=1})#

// Create a query out of an array of entities
#html.table(data=entityLoad('User'), attributes={border=1})#
```

**addStyleContent**

Open and close xhtml style tags so you can easily just add content

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
content	any	Yes	---	The content to render out

**Examples**

```
#html.addStyleContent("body(padding:0;margin:10px)")#
//produces
<style type="text/css">body{padding:0;margin:10px}</style>
```

**link**

Create link tags, using the SES base URL or not

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
href	string	Yes	---	The href link or a structure of name-value pairs to render: keys [href,rel,type,title,media]
rel	string	No	stylesheet	The rel attribute
type	string	No	text/css	The type attribute
title	string	No		The title attribute
media	string	No		The media attribute
noBaseURL	boolean	No	false	Prepends the setting of baseURL if false

**Examples**

The *href* attribute can be a single location or a structure of name-value pairs that simulates the function's arguments:

```
#html.link('includes/tyels.css')#
// produces <link href="http://mysite.com/includes/styles.css" rel="stylesheet" type="text/css" />
#html.link(src="includes/styles/my.css",noBaseURL=false)#
// produces <link href="includes/styles/my.css" rel="stylesheet" type="text/css" />
#html.link(href='includes/syyles.css',media="print")#
// produces <link href="http://mysite.com/includes/styles.css" rel="stylesheet" type="text/css" media="print"/>
```

You can also use the *href* argument as a cool structure of name value pairs that match the arguments in the function:

```
#html.link({href="includes/styles.css",media="print"})#
// produces <link href="http://mysite.com/includes/styles.css" rel="stylesheet" type="text/css" media="print"/>
```

**br**

Generate line breaks

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
count	numeric	No	1	The number of breaks

**Examples**

```
#html.br(3)#
//produces <br/><br/><br/>
```

**heading**

Generate header tags

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
title	string	Yes	---	The header content
size	numeric	No	1	The header size: h1, h2, hx

**Examples**

```
#html.heading("My Awesome Title")#
// produces <h1>My Awesome Title</h1>
#html.heading("My Awesome Title",2)#
// produces <h2>My Awesome Title</h2>
```

**addAsset**

Add a js/css asset(s) to the html head section. You can also pass in a list of assets via the asset argument to try to load all of them. You can also make this method return the string that will be sent to the header instead.

**Returns**

- This function returns *any*

**Arguments**

Key	Type	Required	Default	Description
asset	any	Yes	---	The asset(s) to load, only js or css files. This can also be a comma delimited list.
sendToHeader	boolean	No	true	Send to the header via htmlhead by default, else it returns the content

## Examples

```
// Load to the <head> assets without repeating them during a single request
addAsset("includes/js/jquery.js,includes/js/myJS.js,includes/styles/mystyles.css");
addAsset("includes/js/jquery.js, includes/styles/awesome.css");
// Will still only include the assets that have not been already declared before
```

## meta

Helps you generate meta tags

### Returns

- This function returns *any*

### Arguments

Key	Type	Required	Default	Description
name	any	Yes	---	A name for the meta tag or an array of struct data to convert to meta tags. Keys [name,content,type]
content	any	No		The content attribute
type	string	No	name	Either <i>name</i> or <i>equiv</i> which produces http-equiv instead of the name

### Examples

```
#html.meta("description","My awesome site")#
// Produces
<meta name="description" content="My awesome site">

// Note the third parameter. Can be "equiv" or "name"
#html.meta('Content-type', 'text/html; charset=utf-8', 'equiv')#
// produces
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />

#html.meta(['name' = 'robots', 'content' = 'no-cache']);
// produces
<meta name="robots" content="no-cache" />

// You can also use an entire array of structs
meta = [
{name="robots", content="no-cache"},
{name="description",content="my awesome site"},
{name="content-type", content="text/html; charset=utf-8", type="equiv"}
]

#html.meta(meta)#
// Produces:
<meta name="robots" content="no-cache" />
<meta name="description" content="my awesome site" />
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
```

## nbs

Generate non-breaking spaces ( )

### Returns

- This function returns *any*

### Arguments

Key	Type	Required	Default	Description
count	numeric	No	1	The number of spaces

### Examples

```
#html.nbs(3)#
// produces
```

## addJSContent

Open and close xhtml javascript tags so you can easily just add content

### Returns

- This function returns *any*

### Arguments

Key	Type	Required	Default	Description
content	any	Yes	---	The content to render out

### Examples

```
#html.addJSContent("function myAlert(){ alert('awesome'); }function slide(){ $('#mydiv').slideUp(); }")#
// produces
<script type="text/javascript"><![CDATA[
function myAlert(){ alert('awesome'); }
function slide(){ $('#mydiv').slideUp(); }
]]></script>
```

## docType

Render a doctype by type name: xhtml11,xhtml1-strict,xhtml-trans,xhtml-frame,html5,html4-strict,html4-trans,html4-frame. The default is html5.

### Returns

- This function returns *any*

### Arguments

Key	Type	Required	Default	Description
type	string	No	xhtml1-trans	The doctype to generate

## Examples

```
#html.doctype()#  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
  
#html.doctype('html4-trans')#  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

The available doc types can be seen in the code below:

```
'xhtml11' : '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
'xhtml11-strict' : '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">  
'xhtml11-trans' : default : '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11-transitional.dtd">  
'xhtml1-frame' : '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11-frameset.dtd">  
'html5' : '<!DOCTYPE html>  
'html4-strict' : '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  
'html4-trans' : '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
'html4-frame' : '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
```

## More From The Community

- [Blog: A Look at ColdBox Resource Management](#) by Aaron Greenlee [Sep 14, 2010]