Clickability
by upland

Clickability Platform

Template Language Reference Guide

February 2015

Clickability by upland

Table of Contents

Introduction 8
Welcome 8
Language Organization 8
Using This Guide 9
General Language Reference 10
Language Syntax 10
Tags 10
Comments 11
Variable References 12
Using Set 13
Using Quotes in Velocity Expressions 15
Loops 16
Conditional Statements 17
Parentheses and Spaces 19
Objects 21
Types of Objects 21
Basic Types 21
Clickability Platform Objects 21
Manipulating and Displaying Objects 22
Getting the Object Type 22
Checking for NULL Values 23
Strings 23
Usage of functions 24
Examples: 27
Concatenating Strings 28
Dates 29
Creating Date Objects 29
Date Objects elements 30
Relative Date Creation 31
Date Comparison 32
Date Conversion 34
Date Math 34
Formatting Dates for Display 35
Formatting Dates for Region/Locale 39
Numbers and Math 41
Number Assignment 41

Basic In-line Operations ... 43
Math Operators ... 43
Sum and Average ... 45
Minimum and Maximum ... 46
Formatting Numbers for Display ... 46
Format String Symbols ... 49
ArrayLists ... 50
Creating an Array List ... 50
Adding, Retrieving and Removing Elements ... 50
Getting Information about the ArrayList ... 52
Sorting Array Lists ... 52
HashMaps ... 56
Creating a HashMap ... 56
Adding, Retrieving and Removing Elements ... 56
Getting Information about the Object ... 57
User Defined Objects ... 58
Creating an Object ... 58
Adding Elements ... 58
Retrieving Elements ... 59
Getting Information about the Object ... 61
Using Sub-Templates and Include Files ... 62
Using Sub-Templates ... 62
Accessing Variables in Sub-Templates ... 63
Retrieving Template Data ... 63
Include Files ... 64
Retrieving Include Data ... 65
Content Tags ... 66
Content Item Elements ... 66
Media within Content Items ... 71
Calendar Schedules on Content ... 73
Categorization of Content Items ... 76
Accessing Content Language Information ... 77
Accessing Deleted Content ... 78
Comments ... 79
Basic Comments ... 79
Ratings Comments ... 84
Threaded Comments ... 90
Comments Search ... 94

Formatting Text **97**
Formatting 97
Paragraphs 98
Pagination 99
Translating text 100
Encoding Options 102
Using Placement Lists **108**
Getting Placement Lists 108
Filtering Placement Lists on Content Type 110
Using Categories in Placement Lists 111
Advanced Placement Lists 113
Categories **116**
Getting the Categorization of a Content or Media Item 116
Defining Categorization Criteria 117
Creating a Category Histogram 119
CategorySet Object 121
Category Object 122
Media **123**
Media Items 123
Image Tag Utility 125
Resizing Image Tag Utility 127
Retrieving Secure URLs 129
Destinations **130**
Newsletters (v2) **133**
Newsletter Elements 133
Mailing Elements 134
Mailing List Elements 135
Mailing List Subscription Elements 136
Subscriber Elements 136
Special Newsletter Usage of Standard Site Publishing Tags 138
Sent Mailings 138
Publishing Information **139**
Request Information **141**
URL Information 142
Request Parameters 143
Polls **145**
Using Polls 145
Surveys (version 1) **146**

Surveys (version 2) **147**
Producing an HTML Survey Form 147
Applying Styles to the HTML Survey Form 147
Using your own custom HTML 151
Working with Survey Result Data 152
Working with Survey Response Aggregate Information 153
Calendars **154**
Interactive Tools Integration **155**
Standard SAVE THIS, EMAIL THIS, PRINT THIS, MOST POPULAR & RSS Buttons 155
Custom SAVE THIS, EMAIL THIS, PRINT THIS, MOST POPULAR & RSS Buttons 156
Configure Print This Page Display 157
MOST POPULAR results 158
Tracking **160**
Page Tracking 160
URL Tracking 160
Interactive Tools Tracking 162
Sending Mail **163**
Create a new mail object 163
Add the To and From Addresses: 163
Construct the Mail Message 164
Send the Email 164
Content Submissions **165**
Creating a web submission form 165
Including Media files in a web submission form 169
Constructing a Submission Notification Template 170
Feeds **172**
Retrieve the RSS Feed 172
Display Channel Elements 173
Displaying Feed Items 173
Generating Feeds 173
Other XML Documents 176
Site Subscriptions **177**
Subscribed User Information 177
General Subscription Process Tags 177
Retrieve Password Email 178
Registration Form Helpers 178
Secure Media URLs 178
Registration via Social Media SSO 179

Site Search **180**
Content Site Search tags 180
Media Site Search tags 183
Performing a Site Search Query 186
Displaying Site Search Results 193
Site Search Examples: 195
'More Like This' Site Search 197
Ad Tags **201**
Using Ad Tags 201
Working with Text Ads 203
Flash Ad Tracking 204
Limelight Video Portal (LVP) Integration **205**
Third Party Integrations **207**
Inform 208
AWS SimpleDB 209
Amazon RDS 210
SOAP Messaging from Templates 212
Template-based JSON Parser 218
Web Marketing Acceleration (WMA) **219**
WMA Reference Materials 219
WMA Configuration 219
Progressive Profiles 219
Segmentation 222
Visitor Data 224
Account Data 228
Visitor Lead Score 230
Interests 230
Actions 232
Content and Media 233
Suggested Content Based on Interests 234
Search 235
Campaign Tracking 235
Publisher Tools 236
WMA Code Samples 242
Miscellaneous Tags **245**
HTTP Client Service 245
Access to Outside URLs 247
Remote Server Manager 247
Redirect via Velocity 251

Visitor Location Data Based on IP Address 251
Translation Management using Content Hierarchies 251
Non-Translation Content Hierarchies 255
"In Context" Editing Tags 255
Publisher Encryption Utilities 258
Print to PDF Tags 261
Assorted Tags 262
Velocity Macros (Velocimacros) 262
Debugging Tools 264
Troubleshooting Modes 264
Dynamic Error Pages 267
Performance 268
Caching 268
Robots 269
Connections to other Servers 269
High Load Operations 269
Online Resources 270
Appendix A 271
Appendix B 273
List of All Locales 273
List of All Time Zones 275
Time Zone/ID 275
Acknowledgements 280

Introduction

Welcome

This reference guide details the syntax and usage of the template language used in the Clickability Platform web content management system (WCM). We think you'll find that the Clickability Platform template language is a powerful tool that works well delivering your content because of both its power and simplicity.
The Clickability Platform template language is a templating language that derives many of its characteristics from the underlying Clickability Platform, a java application. As a result, the object oriented approach and general language usage will be very familiar to anyone with java programming experience, although no programming experience is required.

Language Organization

The template language is organized into distinct libraries of tags, each with its own particular purpose.

Library	Tag Prefix	Usage
Content Management	$cms	Interfacing with the Clickability Platform content management system elements, including content items, website section items, templates, media, etc.
Utility	$util	Creating, manipulating and displaying base objects (string, dates, numbers, etc)
String	$string	Working with string elements
Math	$math	Performing arithmetic and other math functions
SQL	$sql	Interfacing with a relational database external to the Clickability Platform system.
Request	$req	Retrieving information specific to the page request being published
Interactive Tools	$imware	Integrating the Clickability Platform Interactive tools into a site published by Clickability
Subscription	$subscription	Working with the Clickability Platform subscription module

Using This Guide

Conventions used in this guide

Several document conventions are used in this guide to help you identify and find information. Typographical conventions are used to indicate specific information types in the documentation.

Convention	Usage in documentation
Bold	This style indicates menu items and buttons that must be clicked or selected. For example, "Once you have completed the form, click **Finish** to move to the next screen."
Italic	This style indicates the title of a book or manual. Most commonly it is used to refer to other documentation in the Clickability Platform information product suite.
Gray- Background	Text with a gray background denotes syntax definition boxes for Clickability Platform template tags and commands.
Code Font	This style denotes source code and names of programming functions. For example, The title is <B>good</B>
Web Output Font	This style denotes output of programming functions, displayed as if in a web browser. The title is **good**

Additional materials

- Community Web site – community.clickability.com – provides additional information, best practices, code samples and an FAQ.
    - Join our Developer Forum!

      Navigate to community.clickability.com > Developer Forum and click "Subscribe" in the right sidebar.
- Corporate Web site - www.clickability.com - provides information about additional features available in the Clickability Platform and other related products such as interactivity and adaptation tools.

General Language Reference

Language Syntax

Tags

The Clickability Platform Template Language tags allow for dynamic content to be embedded into a template (such as a web page HTML template). In their simplest form, these tags are references to elements of saved content, attributes of the website, and the basic information needed to construct a complete web page. Beyond basic use, Clickability Platform template tags provide a powerful means for retrieving content and media from the content management system, manipulating data, constructing websites, and providing mechanisms for interaction with site users.

Tags come in three core types: References, Directives, and Methods.

- **Reference Tags** – refer to variables, or content and media elements, and inserts data into the page
- **Directive Tags** – statements that tell the publisher what to do; includes conditional statements and loops.
- **Method Tags** – tags that perform actions, such as manipulating a string or some sort of external data.

This reference guide details the syntax and uses of all Clickability Platform Template language tags.

Comments

Single Line:

```
## This is a single line comment.
```

Multiple Line:

```
#*
This is a multiline comment.
This is the second line
*#
```

Comments allows descriptive text to be included that is not processed by the template engine. Comments are a useful way of reminding yourself and explaining to others what your statements are doing, or any other information you find useful. Below is an example of a comment:

```
## This is a single line comment.
```

A single line comment begins with ## and finishes at the end of the line. If you're going to write a few lines of commentary, there's no need to create numerous single line comments. Multi-line comments, which begin with #* and end with *#, are available to handle this scenario.

```
This is text that is outside the multi-line comment.  Online visitors can
see it.
#*
 This is a multi-line comment. Online visitors won't
 see this text because the template engine will
 ignore it.
*#
Here is text outside the multi-line comment; it is visible.
```

Here are a few examples to clarify how single line and multi-line comments work:

```
This text is visible. ## This text is not.
This text is visible.
This text is visible. #* This text, as part of a multi-line comment, is not
visible. This text is not visible; This text is still not visible. *# This
text is outside the comment, so it is visible.
## This text is not visible.
```

Variable References

The most basic template tag is a reference to a variable or other piece of data. This variable could be defined in the template itself, retrieved from the content management system, or from a variety of other sources.

For example, the following tags define a variable “color”, set its value to “red”, and place this value into the template in bold face:

```
#set($color = "red")
The color is: <b>$color</b>
```

The output is:

- The color is: **red**

Variable names must start with the “$” character, then an alphabetic character (a .. z or A .. Z). The rest of the characters are limited to the following types of characters:

- alphabetic (a .. z, A .. Z)
- numeric (0 .. 9)
- hyphen ("-")
- underscore ("_")
- Variable names are case sensitive, so $color and $Color are not references to the same variable,

Formal Reference Notation – using {braces}

The above syntax is a short reference notation and is applicable in most cases. However, in some cases it is necessary for a more specific notation, called formal reference notation, to achieve the desired results.

Consider the example of creating an adjective and then wishing to make it the superlative for of that adjective by appending “est” to the end (ie. “small” turns into “smallest”). Using the standard reference notation,

```
#set($word = "small")
The superlative of "$word" is "$wordest".
```

would result in:

- The superlative of “small” is “$wordest”.

The template engine is actually interpreting “$wordest” as a reference to a variable named “wordest”, instead of the variable “word” followed by the extension “est”. In this case, using the formal reference notation by encapsulating the variable name in {braces} is necessary:

```
#set($word = "small")
The superlative of "$word" is "${word}est".
```

would result in:

- The superlative of “small” is “smallest”.

Quiet Reference Notation– using !

If a reference is to an unknown variable, an unknown tag, or a variable with no data, the publishing engine will simply write the unkown tag into the template. For example:

```
#set($color = "red")
The color is: <b>$color</b>
The color is: <b>$colro</b>
The size is: <b>$size</b>
```

will output:

- The color is: red
- The color is: $colro
- The size is: $size

Since these display results would be undesirable and perhaps even detrimental, it is recommended to use quiet reference notation to suppress these invalid, or empty, references. Quiet reference notation places an exclaimation point (!) immediately following the $ in the tag. Using the same example, but with quiet notation:

```
#set($color = "red")
The color is: <b>$!color</b>
The color is: <b>$!colro</b>
The size is: <b>$!size</b>
```

will output:

- The color is: red
- The color is:
- The size is:

Using Set

```
#set( $variable-name = expression )
```

Usage:

- *$variable-name* - The variable being assigned a value.
- *expression* - The value(s) being assigned to the variable. This may be any of the following:
  - Variable reference
  - String literal
  - Property reference
  - Method reference
  - Number literal
  - ArrayList
  - Arithmetic Expression

Variable names must start with the symbol “$” followed by an alphabetic character (a .. z or A .. Z). The rest of the characters are limited to the following types of characters:

- alphabetic (a .. z, A .. Z)
- numeric (0 .. 9)
- hyphen ("-")
- underscore ("_") .

Variable Reference

The set directive can be used to assign the value to a variable from an existing variable.

```
#set($name = $username)
```

String literal

Variables can be assigned string values using double quotes:

```
#set($name = "Mickey Mouse")
```

Number literal

Variables can be assigned numeric values:

```
#set($size = 15)
```

Property reference

When referencing specific pieces of complex object, such as a content item, the individual elements are referenced using property reference syntax:

```
#set($title = $content.title)
#set($summary = $content.summary)
```

Method Reference

The set directive can reference methods and other tags to construct the value to assign to the variable:

```
#set($nocap = $string.toLowerCase($content.title))
```

In this example, the variable “nocap” is assigned the result of passing the “content.title” element through the toLowerCase method (string methods are detailed in a later chapter).

ArrayList

Array lists are sets of values assigned to a single variable. They can be created using “set” or retrieved through several other Clickability Platform template tags to serve a variety of purposes. To create an array list using “set”, the values are assigned in a bracket list of string literals:

```
#set($colors = ["red","blue","green"])
```

Array lists are particularly powerful when used within #foreach loops, as described in the next section of this chapter.

Arithmetic Expression

Arithmetic expressions may also be used in variable assignments (provided both operans are number literals or number literal expressions):

```
#set($a = $b + 1)
#set($a = $b - 1)
#set($a = $b * 2)
#set($a = $b / 2)   (returns only un-rounded whole number)
#set($a = $b % 2)   (modulus operator–returns only the remainder)
```

where $b has a number literal value.

Using Quotes in Velocity Expressions

Velocity expressions enclosed inside of single quotes (or a set of quotes where the outermost pair are single quotes) DO NOT get parsed, so make sure if you have a Velocity expression contained within quotes or a nested set of alternating quotes, that the outermost set of quotes are double quotes.

Examples:

The following code:

```
#set($author = "Jane")
<b>$author</b>
```

…and:

```
#set($author = 'Jane')
<b>$author</b>
```

…both properly output:

- Jane

Whereas:

```
#set($byline = "By $author")
<b>$byline</b>
```

…properly outputs:

- By Jane

while:

```
#set($byline = 'By $author')
$byline
```

…fails to translate the Velocity, outputting instead:

- By $author

Similarly, the $width expression below will not translate here:

```
$util.tag($photo,false, 'width="$width" border="1"')
```

But the $width expression below will translate:

```
$util.tag($photo,false, "width='$width' border='1'")
```

Loops

```
#foreach( $item in $itemList )
[ statement(s)... ]
#end
```

Usage:

- `$item` – The variable name to be used to refer to the current item in the loop. (On the first time loop, $item will refer to the first item in the list, on the next loop to the second, and so on).
- `$itemList` – The list containing the items to be looped through.
- `Statement(s)` – The code to execute each time Velocity finds a valid item ($item) in the list ($itemList).

The #foreach directive allows for looping over lists of items (e.g., ArrayLists), performing the indicated series of statements on each loop.

Loop Examples:

The following code:

```
<ul>
#foreach( $person in $cms.content.authors )
      <li>$person</li>
#end
</ul>
```

would output:

- Alice Author
- Warren Writer
- Ricky Reporter

This #foreach loop causes the “cms.content.authors” list to be looped over for all of the authors in the list. Each time through the loop, the next value from “cms.content.authors” is placed into the $person variable.
The contents of the $cms.content.authors variable is an array of either names or complex content items. As a more complex example, consider a set of baseball games scores:

```
#foreach($score in $cms.content.boxScore)
      $score.homeTeam <b>$score.homeScore </b><br/>
      $score.awayTeam <b>$score.awayScore </b><br/>
      <br/><br/>
#end
```

which would output:

- RedSox 12
- Yankees 3

- San Francisco 9
- Dodgers 2

The template engine provides an easy way to get the loop counter so that you can do something like the following:

```
#foreach($person in $cms.content.authors)
	$velocityCount: $person<br/>
#end
```

which would output:

- 1: Alice Author
- 2: Warren Writer
- 3: Ricky Reporter

Note: there was support built into the Clickability Platform for the Apache VelocityTools "Iterator" Class which has since been deprecated; use $velocityCount instead.

Conditional Statements

```
#if( [condition] )
	[output]
[ #elseif( [condition] )
	[output] ]*
[ #else
	[output] ]
#end
```

Usage:

- **condition -** If a boolean, considered true if it evaluates to true; if not a boolean (e.g., a variable referring to a content object *) considered true if the object (or property) referenced exists.
- **output**–the code to execute when the condition is met. May contain VTL statements.

In a simple conditional statement, the template code between the #if and the #end will be output if the conditional expression is matched:

```
#if($color == "red")
	Yes! It is red.
#end
```

would output:

- Yes! It is red.
  - if $color is "red," and nothing if $color is not "red"

Adding an else part of the conditional statement provides output if the conditional expression is not met.

```
#if($color == "red")
Yes!  It is red.
#else
      No.  It is not red.
#end
```

would output:

- Yes! It is red.
  - If $color is equal to “red”

and

- No. It is not red.
  - if $color is not equal to“red”.

Furthermore, by using #if / #elseif / #end statements, extensive case matching can be accomplished:

```
#if($color == "red")
      It is red.
#elseif($color == "blue")
      It is blue
#elseif ($color == "green")
      It is green
#else
      Color is unknown.
#end
```

would output one of the above lines depending on the value of the variable $color.

Relational Operators

Many relational operators are available for constructing logical expressions for use with the “if” statement. Variable comparisons can be any of the following:

Equivalent Operator:

```
#if( $var1 == $var2 )
```

Greater Than:

```
#if( $var1 > 42 )
```

Less Than:

```
#if( $var1 < 42 )
```

Greater Than or Equal To:

```
#if( $var1 >= 42 )
```

Less Than or Equal To:

```
#if( $var1 <= 42 )
```

Equals Number:

```
#if( $var1 == 42 )
```

Equals String:

```
#if( $var1 == "hello" )
```

Note: When comparing variables in a conditional statement, they must be of the same data type, or the comparison will not work. Date Objects (discussed in the "Dates" chapter) may only be compared with other Date Objects. They may not be compared with strings.

Additionally, as will be discussed in the "Content" chapter, dropdown select list values cannot be compared using "==". Instead, they must use the $string.equals() method.

Logical Operators

Logical operators allow for the construction of more complex conditional expressions by providing the logical operators for AND, OR, and NOT.

AND:

```
#if($var1 == 42 && $var2 == 14)
```

OR:

```
#if($var1=="hello" || $var1=="hi")
```

NOT:

```
#if(!$var1)
```

Using the logical NOT requires that $var1 be a Boolean variable.

Parentheses and Spaces

You will often need to use parentheses in the Clickability Platform Template Language. They provide both structure for syntax, and a mechanism for passing specific parameters to a tag or function.
The following guidelines should be followed to prevent syntax errors and other undesirable results.

Do not place spaces before the open parenthesis when it is part of a tag or needed for syntax:

Bad:

```
#set  ($a = "banana")

#foreach  ($a in [1..9])

#if  ( $a == $ b )
```

Good:

```
#set($a = "banana")

#foreach($a in [1..9])
```

```
$cms.template("topNav")

#if( $a == $b )
```

Do not place any spaces within an individual tag/expression:

Bad:

```
$cms.media(12) .title

$util. now
```

Good:

```
$cms.media(12).title

$util.now
```

You may place spaces within the clauses of #if, #set and #foreach statements to make them easier to read. But any individual tag/expression within the clause must be free of spaces:

Bad:

```
#set( $a = $cms.links (2,5)  )
```

Good:

```
#set(  $a = $cms.links(2,5)  )

#if($a==$b)

#if(  $a   ==   $b   )

#set(  $a=  "abc")
```

Objects

Types of Objects

Every variable or dynamic element is contained within an object. There are many different types of objects, from the very simple (ie. integer) to the complex (ie. a content item). Many of these object types are inherited directly from the java programming language, while others are special interfaces to data elements specific to the Clickability Platform system.

Basic Types

Many basic variable or object types will be very familiar – regardless of experience level in working with programming languages. There are types directly inherited from the Java programming language that provide the basic building blocks of the template language.

Some examples:

Type	Example
Integer	15
Double	-98.472
ArrayList	[12,34,13]
String	"Banana Split"
Boolean	true / false
Date	3/19/2003 11:56:17 AM

Note: This is not a complete list of all object types.

Clickability Platform Objects

Clickability provides many object types (and interfaces to object types) that are specific to the Clickability Platform. These objects are typically composed of multiple component objects and are of varying complexity.

Some examples:

Type	Contains
Content Item	Title Author Name Create Date Modified Date Body Etc.
Website Section	Name Path

	Etc.
PollResults	All the information required to display a poll results page
UserObject	A custom created object containing any number of elements of varying types See the “User Defined Objects” chapter for details

Note: This is not a complete list of all object types.

Manipulating and Displaying Objects

Each type of object is manipulated and displayed in its own way. It is very important to take the correct approach and use the correct syntax to manipulate and display objects, or the desired results may not be achieved. Many types, such as strings, may be simply displayed through a reference tag. However, some objects, like dates, require special tags to ensure the date is displayed in the correct format.

In this guide, syntax definition boxes indicate the object types returned by tags, as well as the object types required by certain methods.

Getting the Object Type

```
String $util.type(object o )
String $util.className(object o)
```

Usage:

- **o** – Any object (string, integer, number, content item, media object, date, etc)

The $util.type tag is used to get the type name of any object. Possible results include:

- Integer
- String
- ArrayList
- Boolean
- UserObject
- HashMap
- Double
- News Article (an example content type name)
- Web_Site_Section (an example website section type name)

The $util.className tag gives you the Java Class of the object in question.

Checking for NULL Values

Boolean

```
$util.isNull(object o )
```

Usage:

- ***o*** – Any object (string, integer, number, content item, media object, date, etc)

The isNull tag can be used for detecting NULL objects. For example:

```
#if($util.isNull($cms.field))
Do one thing
#else
       Do a different thing
#end
```

Strings

String	**$string.charAt**(string *str*, int *i*)
String	**$string.substring**(string *str*, int *startIndex*)
String	**$string.substring**(string *str*, int *startIndex*, int *endIndex*)
String	$string.toLowerCase(string *str*)
String	$string.toUpperCase(string *str*)
String	**$string.trim**(string *str*)
Integer	$string.hashCode(string *str*)
Integer	**$string.indexOf**(string *str*, string *substr*)
Integer	**$string.indexOf**(string *str*, string *substr*, int *startIndex*)
Integer	**$string.lastIndexOf**(string *str*, string *substr*)
Integer	**$string.lastIndexOf**(string *str*, string *substr*, int *startIndex*)
Integer	$string.length(string *str*)
Boolean	**$string.endsWith**(string *str*, string *suffix*)
Boolean	**$string.equals**(string *str1*, string *str2*)
Boolean	**$string.equalsIgnoreCase**(string *str1*, string *str2*)
Boolean	**$string.startsWith**(string *str*, string *prefix*)
String	**$string.replaceAll**(string *str*, string *replace*, string *replacewith*)
String	**$string.replaceFirst**(string *str*, string *replace*, string *replacewith*)
ArrayList	**$string.split**(string *str*, string *separatorRegex*)
ArrayList	**$string.splitString**(string *str*, string *separator*)
ArrayList	**$string.splitString**(string *str*, string *separator,* boolean *trimResults,* boolean *omitEmptyStrings*)
String	**$string.splitOnRegex**(string *str*, string *separatorRegex*)

String	**$string.splitOnRegex**(string *str*, string *separatorRegex,* boolean *trimResults,* boolean *omitEmptyStrings*)
String	$string.removeSpaces(string *str*)

The String functions are all based on the underlying java string methods. Detailed usage is described below:

Usage of functions

```
$string.charAt(string str, int i)
```

- Returns the character at the specified index position (int).

```
$string.substring(string str, int startIndex)
```

- Returns a new string that is a substring of this string, beginning at character position startIndex.

```
$string.substring(string str, int startIndex, int endIndex)
```

- Returns a new string that is a substring of this string, beginning at startIndex and ending at endIndex. Note endIndex is exclusive, meaning the substring returned will not include the character at location endIndex.

```
$string.toLowerCase(string str)
```

- Returns a string with all of the characters in this string converted to lower case.

```
$string.toUpperCase(string str)
```

- Returns a string with a all of the characters in this string converted to UPPER CASE.

```
$string.trim(string str)
```

- Returns a string with white space removed from both ends of this string.

```
$string.hashCode(string str)
```

- Returns a hashcode for this string.

```
$string.indexOf(string str, string substr)
```

- Returns the index within this string of the first occurrence of the specified substring (substr)
- Returns –1 if substr is not found in string.

```
$string.indexOf(string str, string substr, int startIndex)
```

- Returns the index within this string of the first occurrence of the specified substring (substr), starting at the specified index (int).
- Returns –1 if substris not found in string after int.

```
string.lastIndexOf(string str, string substr)
```

- Returns the index within this string of the rightmost occurrence of the specified substring (substr).
- Returns –1 if substr is not found in string.

```
$string.lastIndexOf(string str, string substr, int startIndex)
```

- Returns the index within this string of the last occurrence of the specified substring (substr), starting backwards from index (int).
- Returns –1 if substr is not found in string before int.

```
$string.length(string str)
```

- Returns the length of this string.

```
$string.endsWith(string str, string suffix)
```

- Tests if this string ends with the specified suffix.
- Returns a Boolean value of true or false.

```
$string.equals(string str1, string str2)
```

- Compares whether string is equal to string2.
- Returns a Boolean value of true or false.

```
$string.equalsIgnoreCase(string str1, string str2)
```

- Compares whether string is equal to string2, ignoring case considerations.
- Returns a Boolean value of true or false.

```
$string.startsWith(string str, string prefix)
```

- Tests if this string starts with the specified prefix
- Returns a Boolean value of true or false.

```
$string.replaceAll(string str, string replace, string replacewith)
```

- Returns a string in which all instances of replace-string are replaced with replace-with-string in string.

  Note: "?" and "(" and ")" characters need to be replaced with a backslash "\" such as:

```
#set($regex = '\)')
$string.replaceAll($myString,$regex,") ")
$string.replaceFirst(string str, string replace, string replacewith)
```

- Returns a string in which the first, and only the first, instance of replace-string is replaced by replace-with-string in string.

```
$string.split(string str, string separatorRegex)
```

- Returns a string that is a List of pieces in which string is split based on the delimiter split-on-string.

  Note: This method is still supported, but has been replaced with $string.splitString and $string.splitOnRegex methods below.

```
$string.splitString(string str, string separator)
```

- Returns an array of strings that has been split on every instance of the separator. Results are automatically trimmed of whitespace, and empty string results will be eliminated (this is equivalent to $string.splitString (stringToSplit, separatorString, true, true) detailed below).

```
$string.splitString(string str, string separator, boolean trimResults,
boolean omitEmptyStrings)
```

- Returns an array of strings that has been split on every instance of the separator. If trimResults is true, results are trimmed of whitespace. If omitEmptyStrings is true, result which have a length of 0 will not be returned.

```
$string.splitOnRegex(string str, string separatorRegex)
```

- Similar to the methods above, except that the splitting will occur with the separator treated as a regular expression.

```
$string.splitOnRegex(string str, string separatorRegex, boolean trimResults,
boolean omitEmptyStrings)
```

- Similar to the methods above, except that the splitting will occur with the separator treated as a regular expression. If trimResults is true, results are trimmed of whitespace. If omitEmptyStrings is true, result which have a length of 0 will not be returned.

```
$string.removeSpaces(string str)
```

- Returns a string with all spaces removed from the string string.

Examples:

One Character Split Example

```
#foreach($item in $string.split("abc-def-ghi","-"))
       $item <br/>
#end
```

produces:

- abc
- def
- ghi

Multi-Character Split Example

```
#foreach($item in $string.split("abc-X-def-X-ghi","-X-"))
$item <br/>
#end
```

produces:

- abc
- def
- ghi

Multi-Line String Split Example

Remember to use the escaping backslash when necessary, such as when you are replacing newline characters in a multi-line string:

```
#foreach($stringStripped in $string.splitOnRegex($textToStrip,
"(\\r\\n?|\\n)"))
  <p>String $velocityCount: $stringStipped</p>
#end
```

Simple Regular Expression Split Example

```
#foreach($item in $string.split("abc-1-def-2-ghi-3-jkl","-.-"))
       $item <br/>
#end
```

produces:

- abc
- def
- ghi
- jkl

Concatenating Strings

To concatenate—or join—string values (be they string literals or string expressions), simply enclose them within the same set of quotes.

Example

```
#set($rowPrefix = "Row")
#set($rowLetter = "A")
#set($row = "$rowPrefix: $rowLetter")
```

In this example, $row will then have the value “Row: A”

Note: If the Velocity expression itself contains quotes, they should be depicted as single quotes, to avoid interfering with the outer containing quotes. Example:

```
#set($foo = "Username: $req.param("name")")
```

will not work, because the open double-quotes for “name” is interpreted as a close double-quotes for the “Username… block. Instead, use:

```
#set($foo = "Username: $req.param('name')")
```

or use an intermediate variable, like:

```
#set($var = $req.param("name"))
#set($foo = "Username: $var")
```

Dates

Date objects can be particularly tricky to work with since they don't follow the same rules of comparison, modification, and display as number and string objects. This chapter details the many options available for creating dates, comparing dates, performing date math, and displaying dates.

Creating Date Objects

Date	$util.now
Date	$util.newDate
Date	**$util.newDate**(int *year*, int *month*, int *day*)
Date	**$util.newDate**(int *year*, int *month*, int *day*, int *hour*)
Date	**$util.newDate**(int *year*, int *month*, int *day*, int *hour*, int *minute*)
Date	**$util.newDate**(int *year*, int *month*, int *day,* int *hour*, int *minute*, int *second*)
Date	**$util.parseDate**(string *date*, string *format*)

Usage:

- **format** – The format string for the date. The default format string is "MMM d, yyyy".
- **date** – A date that is a text string (not a date object)
- **year** – An integer representing the year.
- **month** – An integer representing the month [1-12].
- **day** – An integer representing the day.
- **hour** – An integer representing the number hours [0-23].
- **minute** – An integer representing the number minutes [0-59].
- **second** – An integer representing the number of seconds [0-59].

This tag,

```
$util.newDate
```

- will create a **date object** (a date that can be manipulated and compared to other date objects) with the current date and time.

The newDate tag may also be used with parameters to specify an exact date and/or time. The following would create a date object for January 31, 2005:

```
$util.newDate(2005, 1, 31)
```

This tag,

```
$util.parseDate
```

- will create a **date object** (a date that can be manipulated and compared to other date objects) from a text string representing a date.

For example:

```
$util.parseDate("12/12/2012", "MM/dd/yyyy")
```

- will return a **date object** when the first parameter is a date format string representing the date that is to be converted.

The $util.now and the $util.parseDate tags create **date objects** which can be compared with the $compareDates tag. They cannot be compared as **strings**, nor printed to the page as **strings**.

Note: These Date objects **ARE NOT** the same type of object returned by retrieving a date field value from a WCM object such as a content item or website section. If you want to use the Date methods from this chapter with a WCM object date value, you should first use $util.parseDate to cast the WCM date into a Date object, like this:

```
## Assume $cms.content.releaseDate returns "2206-01-26"
#set($releaseDateObject = $util.parseDate("$cms.content.releaseDate", "yyyy-MM-dd"))
```

Date Objects elements

The individual elements of a date can be retrieved with the following underlying java methods. Note the parentheses are required.

Tag	Returns	Example
#set($gameTime = $util.now)		
$gameTime	Entire date/time	Mon Aug 03 17:33:56 PDT 2009
$gameTime.getDate()	Day of the month	3
$gameTime.getDay()	Day of week (Sunday=0, Monday=1, etc.)	1
$gameTime.getMinutes()	Minutes of the hour	33
$gameTime.getHours()	Hour of the day	17
$gameTime.getMonth()	Month of the year (Jan=0, Feb=1, etc.)	7
$gameTime.getSeconds()	Seconds of the minute	56
$gameTime.getTime()	milliseconds since January 1, 1970, 00:00:00 GMT	1249346036485
$gameTime.getTimezoneOffset()	Minutes of offset from GMT	420
$gameTime.getYear()	Years since 1900	109

Relative Date Creation

Date	$util.now
Date	$util.tomorrow
Date	$util.tomorrow(date date1)
Date	$util.yesterday
Date	$util.yesterday(date date1)
Int	$util.currentYear

These special date creation tags allow you to easily create dates that would commonly be referenced, namely 'tomorrow' and 'yesterday'. Because they are relative to the current time, templates using these tags should be cache-purged daily. Ask Client Services about using a special "Refresh Nightly" template to achieve this.

```
$util.tomorrow
```

- Returns a date object representing the day after today.

```
$util.tomorrow(date date1)
```

- Returns a date object representing the day after the day specified by date1.

```
$util.yesterday
```

- Returns a date object representing the day before today.

```
$util.yesterday(date date1)
```

- Returns a date object representing the day before the day specified by date1.

Date Comparison

Date objects **cannot** be compared using the standard greater-then / less-then operators. They must be compared using special date comparison tags.

Boolean	$util.inSameHour(date date1, date date2)
Boolean	$util.inSameDay(date date1, date date2)
Boolean	$util.inSameMonth(date date1, date date2)
Boolean	$util.inSameYear(date date1, date date2)
Boolean	$util.isBefore(date date1, date date2)
Boolean	$util.isAfter(date date1, date date2)
Boolean	$util.isEqualDate(date date1, date date2)
Boolean	$util.isEqualTime(date date1, date date2)
Boolean	$util.isBetween(date date1, date date2, date date3) where date3>date2
Integer	$util.compareDates(date date1, date date2)
Date Compaison Tool	
Date Comparison Object	$dateComparisonTool.whenIs(date *date1,* date *date2*)
Date Comparison Object	$dateComparisonTool.difference(date *date1,* date *date2*

Usage:

- date1 – A date object
- date2 – A date object
- date3 – A date object

`$util.inSameHour(date date1, date date2)`

- Returns true if date1 and date2 are within the same hour. Otherwise it returns false.

`$util.inSameDay(date date1, date date2)`

- Returns true if the date1 and date2 are within the same day. Otherwise it returns false.

`$util.inSameMonth(date date1, date date2)`

- Returns true if the date1 and date2 are within the same month. Otherwise it returns false.

`$util.inSameYear(date date1, date date2)`

- Returns true if the date1 and date2 are within the same year. Otherwise it returns false.

`$util.isBefore(date date1, date date2)`

- Returns true if date1 is before date2. Otherwise it returns false.

```
$util.isAfter(date date1, date date2)
```

- Returns true if date1 is after date2. Otherwise it returns false.

```
$util.isEqualDate(date date1, date date2)
```

- Returns true if the date portion (years, months, date) of date1 equals the date portion of date2. Otherwise it returns false.

```
$util.isEqualTime(date date1, date date2)
```

- Returns true if the entire date and time (years, months, date, hours, minutes, seconds, milliseconds) of date1 equals date2. Otherwise it returns false.

`$util.isBetween(date date1, date date2, date date3)` where date3 > date2

- Returns true if date1 is between date2 and date3 where date3>date2. Otherwise it returns false.

```
$util.compareDates(date date1, date date2)
```

- Returns 1 if date1 is after date2, returns –1 if date1 is before date2 and returns 0 if the dates are identical (year to milliseconds)

Example:

```
#set($now = $util.now)
#set($gameTime = $cms.content.gameTime)
#if($util.compareDates($now,$gameTime) == 0)
       The game starts exactly right now!
#elseif($util.compareDates($now,$gameTime) > 0)
       The game already started.
#elseif($util.compareDates($now,$gameTime) < 0)
       The game has not started yet.
#end
```

Note: compareDates will always compare down to the millisecond as date objects are complete date/time representations.

Date Comparison Tool

The Date Comparison Tool can compare dates and find the relationship between any date and the current date, or between any two dates. This comparison can result in either a textual representation of the relationship (e.g. “3 weeks, 2 days ago”, “tomorrow”, or “3 hours away”) or the value of a specific time unit may be requested.

Note: this tool is part of the Apache Velociy Tools project and has been integrated into the Clickability Platform Template Language. Refer to http://velocity.apache.org/tools/devel/javadoc/org/apache/velocity/tools/generic/ComparisonDateTool.html for more information and available methods.

```
$dateComparisonTool.whenIs(date date1, date date2)
```

- Returns a Date Comparison between the second specified date and the first specified date.

Note:

```
$dateComparisonTool.whenIs($util.newDate(),$util.newDate(yyyy,mm,dd))
```

- would result in “XX years/months/weeks/days earlier/later” based on today’s date, e.g. “15 years earlier” or “2 weeks later”.

```
$dateComparisonTool.difference(date date1, date date2)
```

- Returns a Date Comparison between the result of the second specified date and the first specified date.

Example:

```
$dateComparisonTool.difference($util.newDate(),$util.newDate(yyyy,mm,dd))
```

- would result in “XX years/months/weeks/days” based on today’s date, e.g. “-15 years” or “2 weeks”.

Date Conversion

There are 2 different types of date objects in the Clickability Platform: the type of Date object documented in this chapter’s methods – which has a time component (regular Date), and what is called a ‘Castor Date’ which comes from custom Clickability Platform date fields, and does not have a time component (e.g. $contentItem.endDate). Wherever you see Date as a method parameter, you should be able to use either type of date. However, if you need to convert a Castor Date to a regular Date, use this method. Just be aware that it will assign a time of midnight PST to all Castor dates, unless another timezone is specified.

```
Date   $util.toDateTime(CastorDate date)
Date   $util.toDateTime(CastorDate date, String timezone)

Usage:
date - The Castor Date field you want to convert
timezone -The timezone you want to create the time in
```

Date Math

```
Date   $util.add(date date1, string date-field, int amount)

Usage:
date-field - The name of the piece of the date object to be operated on.
amount - The time zone for which to display the date
```

The $util.add tag provides a mechanism to add or substract from a date object. The date is actually manipulated on a calendar, so such variations such as leapyears and the number of days in each month are accounted for.

The increment by which to change the date (ie. year, month, minute) is specified by the date-field name. Below is a table of the date field names.

Date Field Names

Field Name
Year
Month
Day
Hour
Minute
Second
Millisecond

Examples:

```
#set($myDate = $util.newDate(2008,7,28,8,25,30))
```

Example	Result
$myDate	Mon Jul 28 08:25:30 PDT 2008
$util.add($myDate,"year",4)	Sat Jul 28 08:25:30 PDT 2012
$util.add($myDate,"month",2)	Sun Sep 28 08:25:30 PDT 2008
$util.add($myDate,"day",-4)	Thu Jul 24 08:25:30 PDT 2008
$util.add($myDate,"day",365)	Tue Jul 28 08:25:30 PDT 2009
$util.add($myDate,"hour",-12)	Sun Jul 27 20:25:30 PDT 2008
$util.add($myDate,"year",-4)	Wed Jul 28 08:25:30 PDT 2004

Formatting Dates for Display

String **$util.date**

String **$util.date**(date *date1)*

String **$util.date**(string *format*)

String **$util.date**(string *format,*string *timezone*)

String **$util.date**(string *format*, date *date1*)

String **$util.date**(string *format*, date *date1,* string *timezone*)

Usage:

- **date1** – a date object
- **format** – The format string for the date. The default format string is “MMM d, yyyy”.
- **timezone** – The long name or GMT time zone for which to display the date.

These date functions allow for dates to be published into pages and allows for the comparison and formatting of date objects. A *format string* is constructed using the symbols detailed below. If the date reference parameter is

omitted from the tag, the tag will default to the current Date and time. If the time zone parameter is omitted, it defaults to Pacific Daylight Time.

Note: The short name timezone (PST, EST, MST, etc) is no longer supported as an accurate timezone designator when passed via the $util.date tag. Please use the long name or specific GMT timezone for best results. Refer to Appendix B for a list of acceptable long name values.

Format String Symbols

Symbol	Meaning	Example
G	era designator	AD
y	Year	03
yyyy	Year	2003
M	month in year	1
MM	month in year	01
MMM	month in year	Jan
MMMM	month in year	January
d	day in month	10
h	Hour in am/pm (1-12)	12
H	Hour in day (0-23)	0
m	minute in hour	30
s	second in minute	55
S	Millisecond	978
E	day in week	Tue
EEEE	day in week	Tuesday
D	day in year	189
F	day of week in month	2 (2nd Wed in July)
w	week in year	27
W	week in month	2
a	am/pm marker	PM
K	Hour in am/pm (0-11)	0
k	Hour in day (1-24)	24
z	Time zone	PST
zzzz	Time zone	Pacific Standard Time
Z	Time zone	-0800

'	escape for text	
''	single quote	'

Format String Examples:

Format String	Output
"yyyy.MM.dd G 'at' hh:mm:ss z"	2009.07.10 AD at 12:08:00 PDT
"EEE, MMM d, ''yy"	Fri, Jul 10, '09
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyy.MMMMM.dd GGG hh:mm aaa"	2009.July.10 AD 12:08 PM

Note: Single quotes are used in the above examples to insert a static character into the formatted output.

Time zones

Time zones may be designated in one of three ways:

Short name: PST, MST, EST, etc.

Long name: America/Los_Angeles
Refer to Appendix B for a list of acceptable long name values

Specifically: GMT[+|-]hh[[:]mm], for example GMT-8:00

Date Reference

The date reference may be any valid date object from the content of media selection. Omitting a date object causes the function to use the current date and time.

Display Dates as Strings

```
$util.date
```

- returns a **string** representing the current date and time in the default date/time format:
  - Apr 17, 2002

```
$util.date("hh:mm MM/dd/yyy")
```

- returns a **string** representing the current date and time in the following format:
  - 01:30 05/09/02

```
$util.date("hh:mm MM/dd/yyy","America/New_York")
```

- returns a **string** representing the current date and time in the Eastern Time Zone in the following format:
  - 04:30 05/09/02

```
$util.date("hh:mm", "GMT-8:00")
```

- returns a **string** representing the current date and time in the PST with the following format:
  - 3:19 PM, PDT

```
$util.date("K:mm a", $cms.content.date, "America/Los_Angeles")
```

- returns a **string** representing the “date” element of the content item in the default date format in Pacific Standard Time:
  - Dec 9, 2001

```
$util.date("hh 'o''clock' a, yyyy.MMMMM.dd GGG",$cms.content.date)
```

- returns a **string** representing the “date” element of the content item in format:
  - 12 o'clock AM, 2001.December.09 AD

Formatting Dates for Region/Locale

String $util.getLocale(string localeNameOrCode, string timeZoneId) - this method returns a locale context object which can be used to format dates, times, currency and numbers in the proper locale specific way

Usage:

- **localeNameOrCode** – a string specifying the locale, either by name “English (United Kingdom)” or by code “en_GB”
- **timeZoneId** – a string specifying the time zone which should be used for outputting dates such as “America/New_York or “Europe/London”

Refer to Appendix B for a complete list of locales and time zones.

Locale Context Specification

Date/Time Format Variants - Each locale specific date and time format has 4 format variants, full, long, medium and short. When formatting a date or time, the variant can either be specified, or if left off, the default which is medium will be used.

For example, the following shows date/time combination would be formatted in each variant:

Format Variant	Output Example: (en_US locale, America/Los Angeles time zone)
full	Monday, July 9, 2012 11:02:19 AM PDT
long	July 9, 2012 11:02:19 AM PDT
medium (default)	Jul 9, 2012 11:02:19 AM
short	7/9/12 11:02 AM

Given a Locale Object $regionLocale

String $regionLocale.name

String $regionLocale.code

String $regionLocale.timeZone

String $regionLocale.formatDate(CastorDate *date*)

String $regionLocale.formatDate(CastorDate *date*, string *formatVariant*)

String $regionLocale.formatDate(Date *date*)

String $regionLocale.formatDate(Date *date*, string *formatVariant*)

String $regionLocale.formatDateTime(Date *date*)

String $regionLocale.formatDateTime(Date *date*, string *formatVariant*)

String $regionLocale.formatDateTime(Date *date*, string *dateFormatVariant,* string *timeFormatVariant*)

String $regionLocale.formatTime(Date *date*)

String	$regionLocale.formatTime(Date *date,* string *formatVariant*)
String	$regionLocale.formatCurrency(number)
String	$regionLocale.formatInteger(integer)
String	$regionLocale.formatNumber(number)
String	$regionLocale.formatPercent(number)

Usage:

- **localeNameOrCode** – a string specifying the locale, either by name “English (United Kingdom)” or by code “en_GB”
- **timeZoneId** – a string specifying the time zone which should be used for outputting dates such as “America/New_York or “Europe/London”
- **formatVariant** – type of format; refer to output examples in table above; note the format can differ between date strings and time strings

Notes:

- Castor” date objects are returned from date only fields in content items, and they manage only dates without any time component.
- Currency, Numbers and Percent inputs support int, long, float and double, whereas Integer input supports only int or long.
- Refer to Appendix B for a complete list of locales and time zones, or use the following tags that are intended only as a development tool so that permitted locale and time zone names can be discovered at development time:
- $util.allLocales - Returns an array of all available locales, including both the locale name, and associated locale code, either of which is valid for getting a locale context (but not together as output by this method).
- $util.allLocaleNames - Returns an array of all available locale names, each of which is valid for getting a locale context.
- $util.allLocaleCodes - Returns an array of all available locale codes, each of which is valid for getting a locale context.
- $util.allTimeZones - Returns and array of all available time zone IDs, each of which is valid when getting a locale context.

Usage Examples:

```
#set($caliLocale = $util.getLocale("English (United States)",
"America/Los_Angeles"))
#set($frenchLocale = $util.getLocale("fr_FR", "Europe/Paris"))
#set($now = $util.newDate)

Date (California): $caliLocale.formatDate($now)
Time (California): $caliLocale.formatTime($now)
Date/Time (California): $caliLocale.formatDateTime($now, "full", "full")
Currency (California):
$caliLocale.formatCurrency($math.newDouble("32687.39"))

Date (France): $frenchLocale.formatDate($now)
Time (France): $frenchLocale.formatTime($now)
Date/Time (France): $frenchLocale.formatDateTime($now, "full", "full")
Currency (France): $frenchLocale.formatCurrency($math.newDouble("32687.39"))
```

would output:

- Date (California): Jul 9, 2012
- Time (California): 12:27:37 PM
- Date/Time (California): Monday, July 9, 2012 12:27:37 PM PDT
- Currency (California): $32,687.39
- Date (France): 9 juil. 2012
- Time (France): 21:27:37
- Date/Time (France): lundi 9 juillet 2012 21 h 27 CEST
- Currency (France): 32 687,39 €

These tags can be used for informational needs during development:

```
#foreach($locale in $util.allLocales)$locale<br>
#end <br>
#foreach($locale in $util.allLocaleCodes)$locale<br>
#end <br>
#foreach($locale in $util.allLocaleNames)$locale<br>
#end <br>
#foreach($tz in $util.allTimeZones)$tz <br>
#end <br>
```

Numbers and Math

There are several types of numbers: integers, longs, floats, and doubles. The two most widely used are integers (whole numbers) and doubles (decimal numbers).

In this chapter, an "Integer" or "int" refers to an integer number; a "double" refers to a double number; and, a "number" refers to any number object including integers, doubles, strings representing number ("123.4"), floats, and longs.

Number Assignment

#set($i = 10)

Integer	$math.newInt
Integer	$math.newInt(number *n*)
Double	$math.newDouble
Double	$math.newDouble(number *n*)
Double	$math.random
Float	$math.newFloat
Float	$math.newFloat(number *n*)

Usage:

- The above tags are used to create new number objects.
- In the absence of any parameters, $math.newInt, $math.newDouble, and $math.newFloat will return zero value Integer, Double, or Float objects, respectively.

Note: Decimal values cannot be specified as literals (e.g., 4.55). Instead, they must be specified as strings ("4.55") or first cast into variables.

Creating numbers with a Value

The following tags:

```
$math.newInt(number n)
$math.newDouble(number n)
$math.newFloat(number n)
```

create new Integers, Doubles and Floats based on the value of the integer, double or float represented by the number parameter (n).

Notes:

- The $math.newInt method treats decimal strings as invalid arguments (and hence returns zero), but it does accept decimal numeric variables (i.e., Doubles, Floats) as valid arguments, and simply truncates the decimal portion to return only the whole number. This is a useful way to "floor" or "round down" a number.
- Trailing spaces are automatically trimmed on strings passed into $math.newInt

Examples:

```
$math.newInt("5")
```

- returns an integer of value 5

```
$math.newInt("abc")
```

- returns an integer of value 0 since the string does not represent a valid integer

```
$math.newInt("4.35")
```

- also returns an integer of value 0 since the string does not represent a valid integer

```
#set($doubleVar = $math.newDouble("4.35"))
$math.newInt($doubleVar)
```

- returns an integer of value 4, since when it is passed a valid double variable, it truncates the decimal

```
$math.newDouble("5")
```

- returns a double of value 5.0

```
$math.newDouble("1.35")
```

- returns a double of value 1.35

Obtaining a Random Number

$math.random will return a random decimal value between 0 and 1.

Basic In-line Operations

```
#set($i = j + k)
#set($i = j - k)
#set($i = j * k)
#set($i = j / k)
#set($i = j % k)
```

Note: The above in-line operators are applicable **only to integers**. If you need to perform operations on objects other than integers, you must use the tags in the following section. Also, if either j or k is a variable reference it is recommended to use the math tags in the following section or the desired results may not be achieved.

For example, the above division function will return only the whole number, truncating any remainder, such that if j = 11 and k = 2, j / k would return 5.

Math Operators

Integer	$math.add(int *i*, int *j*)
Double	$math.add(number *x*, number *y*)
Integer	$math.subtract(int *i*, int *j*)
Double	$math.subtract(number *x*, number *y*)
Integer	$math.multiply(int *i*, int *j*)
Double	$math.multiply(number *x*, number *y*)
Double	$math.divide(int *i*, int *j*)
Double	$math.divide(number *x*, number *y*)
Integer	$math.mod(int *i*, int *j*)
Double	$math.sqrt(number *x*)
Double	$math.pow(number *x*, number *y*)
Integer	$math.abs(int *i*)
Double	$math.abs(number *x*)
Integer	$math.round(number *x*)

The above tags are used to perform basic math functions on integers, doubles, and other number objects

Examples:

Operation	Example	Results
Addition	$math.add(10,4)	14
Subtraction	$math.subtract(30, "7.5")	22.5
Multiply	$math.multiply("4","8")	32.0
Divide	$math.divide(10,4)	2.5
Modulus	$math.mod(10,4)	2
Square Root	$math.sqrt(9)	3
Power	$math.pow("4.0",2)	16.0
Absolute Value	$math.abs(-10)	10
Round	$math.round("10.4")	10

Sum and Average

Double	$math.sum(arrayList *a*)
Double	$math.average(arrayList *a*)
Double	$math.weightedAverage(arrayList *b*)

Usage:

- Where a is an arrayList of numbers: ([x1,x2,x3,…,xn])
- Where b is an arrayList of numbers: [x1,y1,x2,y2,x3,y3,…,xn,yn])

Sums, averages, and weighted averages can all be computed on an ArrayList of numbers.

Given an ArrayList of numbers ([x1, x2, x3, …, xn]), the sum is defined as:

```
x1+ x2 + x3 + … + xn
```

and the average is defined as:

```
( x1+ x2 + x3 + … + xn ) / n
```

Given an ArrayList of numbers ([x1, y1, x2, y2, x3, y3, …, xn, yn]), the weighted averafe is defined as:

```
(x1y1 + x2y2 + x3y3 + … + xn) / (x1 + x2 + x3 + … xn)
```

Examples:

Example	Result
$math.sum(10,4)	14
$math.sum(30,”7.5”,10)	47.5
$math.average(“4”,”8”)	6
$math.average(“10.9”, 4)	7.45
$math.weightedAverage([2,”10.9”,1,4])	8.6

Minimum and Maximum

Integer	$math.min(int i,int j)
Double	$math.min(number x, number y)
Integer	$math.max(int i, int j)
Double	$math.max(number x, number y)
Double	$math.min(arrayList a)
Double	$math.max(arrayList a)

Where a is an arrayList of numbers: ([x1,x2,x3,…,xn])

Maximum and Minimum functions are provided for use with 2 integers or numbers, or with an array list full of numbers. The numbers in the array list may even be of different types.

Examples:

Example	Result
$math.min(10,4)	4
$math.min(30, "7.5")	7.5
$math.max("4","8")	8.0
$math.max(10.9, 4)	10.9
$math.min([10,4,"-2"])	-2.0
$math.min([1,1,1,2])	1.0
$math.max(["4.0",2,100,2000])	2000.0
$math.max(["0.1","-0.1"])	0.1

Formatting Numbers for Display

String	$util.formatNumber([string format], number n)
String	$util.formatAsDollars(number n)
String	$util.formatAsMoney(number n)
String	$util.percent(number n)
String	$util.formatAsBytes(number n)
String	$util.formatAsBytes(number n, boolean useAbbreviations)
String	$util.formatAsBytes(number n, boolean useAbbreviations, boolean allLowercase)
String	$util.formatAsBytes(number n, string format, boolean useAbbreviations, boolean allLowercase)

Usage:

- **format** – The format string for the date. The default format string is “000,000”.
- **number n**– An integer (10), double (10.5), or valid string representation of a number (“10”).

Note: refer to the previous “Dates” chapter for formatting tags to display numbers such as currency, integers, numbers, and percents based on region/locale

Aside from the general arithmetic syntax previously presented, there are several functions that allow for the flexible formatting of numbers. A format string is constructed using the symbols detailed below. Additionally, several specific tags are provided for common formatting needs.

Formatting Byte Values

Using the formatAsBytes tag allows you to easily form byte values in a nice format. This tag will automatically detect whether to display the results in Bytes, KiloBytes (>1024 Bytes), or MegaBytes (>1024 KiloBytes). The appropriate unit is included in the string returned according to the table below.

Additionally, the number of decimal places is adjusted according to the magnitude of the value returned. A number less than 10 will display 2 decimal places, a number between 10 and 100 will display 1 decimal place, and all numbers greater than 100 will display with zero decimal places.

useAbbreviations flag	allLowerCase flag	Units Displayed
false	false	• Bytes • KiloBytes • MegaBytes
true	false	• Bytes • KB • MB
false	true	• bytes • kilobytes • megabytes
true	true	• bytes • kb • mb

Examples:

```
$util.formatAsDollars("12345.32")
$util.formatAsDollars("12345")
```

- These tags would output the number in dollars (no cents):
  - $12,345

```
$util.formatAsMoney("12345.54")
```

- This tag would output the number in dollars and cents:
  - $12,345.54

```
$util.formatAsMoney("12345")
```

- This tag would output the number in dollars and cents:
  - $12,345.00

```
$util.percent(".12")
```

- This tag would output the number as a percent:
  - 12%

```
$util.formatNumber("000,000", "123")
```

- would output:
  - 000,123

```
$util.formatNumber("*.000", "123")
```

- would output:
  - 123.000

```
$util.formatNumber(",*** ; [,***]","123456")
```

- would output
  - 123,456

```
$util.formatNumber(",*** ; [,***]","-123456")
```

- would output:
  - [123,456]

See below for more on using format strings with the formatNumber tag.

Format String Symbols

Symbol	Location	Meaning
0	Number	Digit, zero displayed as a placeholder for missing digits
*	Number	Digit (note: in JAVA and other programming languages and applications, a digit is typically represented by a #)
.	Number	Decimal separator or monetary decimal separator
-	Number	Minus sign
,	Number	Grouping separator
E	Number	Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix.
;	Subpattern boundary	Separates positive and negative subpatterns
%	Prefix or suffix	Multiply by 100 and show as percentage
0	Number	Digit

Format String Examples:

Format String	Applied to:	Output
"***,***"	12345	12,345
"000,000"	12345	012,345
"**%"	0.12	12%
"0.***E0"	12345	1.234E3
"*.00"	1234.5	1234.50

For more info: the specification for formatting numbers and decimal format strings can be found in Java 2 platform documentation for the class DecimalFormat at:

http://java.sun.com/j2se/1.4/docs/api/java/text/DecimalFormat.html

ArrayLists

ArrayLists are data structures that hold a set of objects in a specific order. These individual objects can be accessed by their respective index numbers and may be of any object type, or combination of object types.

ArrayLists are "zero-indexed" meaning that the first element is located at position zero.

Given the ArrayList:

```
["one", "two", "three", "four"]
```

The elements are indexed as:

Element	"one"	"two"	"three"	"four"
Index	0	1	2	3

Creating an Array List

```
ArrayList          $util.newArrayList
```

The $util.newArrayList tag creates a new empty ArrayList object. Once created, elements can be added to it.

Adding, Retrieving and Removing Elements

#set($myArrayList = $util.newArrayList)

Boolean	$myArrayList.add(object *o*)
Boolean	**$myArrayList.add**(int *index*, object *o*)
Boolean	$myArrayList.addAll(ArrayList *a*)
Boolean	**$myArrayList.addAll**(int *index*, ArrayList *a*)
Object	**$myArrayList.set**(int *index*, object *o*)
Object	$myArrayList.get(int *index*)
Object	$myArrayList.remove(int *index*)
ArrayList	**$myArrayList.subList**(int *startIndex*, int end*Index*)
	$myArrayList.clear()

Usage:

- **index** – The numeric position in the array, with 0 being the first
- **list-size** – The number of items to return
- **startIndex** – The index of the item at which to start the range (inclusive)
- **endIndex** – The index of the item at which to end the range (exclusive)
- **a** – Any ArrayList object
- **o** – Any object (string, integer, number, content item, media, object, date, etc with the exception of ArrayLists.)

Adding Elements

```
$myArrayList.add(object o)
```

- This tag will add the object o to the end of the ArrayList. Note: adding an object to the array using this syntax causes “true” to be displayed on the page; one way to avoid this is to use #set($foo=$myArrayList.add(object o)).

```
$myArrayList.add(int index, object o)
```

- This tag will insert the object o to the ArrayList in the position of the specified index value, with all elements after that index value being bumped one position higher. The element will not be added if the index value specified is less than 0 or greater that the size of the ArrayList.

```
$myArrayList.addAll(ArrayList a)
```

- This tag will add all the elements of ArrayList a to the end of ArrayList.

```
$myArrayList.addAll(int index, ArrayList a)
```

- This tag will insert all the elements of ArrayList a to ArrayList in the position of the specified index value, with all elements after that index value being bumped the appropriate position higher. The elements will not be added if the index value specified is less than 0 or greater that the size of the ArrayList.

Replacing an Element

```
$myArrayList.set(int index, object o)
```

- This tag will replace the element at the specified index value with object o. The object that currently resides at this location is returned.

Retrieving Elements

```
$myArrayList.get(int index)
```

- This tag will return the element at the specified index value. It will return null if the index value specified is less than 0 or greater that the size of the ArrayList

```
$myArrayList.subList(int startIndex, int endIndex)
```

- This tag will return a subset of the ArrayList, starting with (and including) the element at the specified startIndex value, and stopping at (i.e., excluding) the specified endIndex value, as long as the ArrayList size is equal or larger than the endIndex value (if it is smaller, nothing will be returned).

Removing Elements

```
$myArrayList.remove(int index)
```

- This tag will remove the element at the specified index value (and return it to you). All elements after that index in the array will be shifted one position lower.

```
$myArrayList.clear()
```

- This tag removes all elements from the array.

Getting Information about the ArrayList

Boolean	$myArrayList.contains(object o)
Integer	$myArrayList.indexOf(object o)
Boolean	$myArrayList.isEmpty()
Integer	$myArrayList.lastIndexOf(object o)
Integer	$myArrayList.size()

Usage:

- **o** – Any object (string, integer, number, content item, media, object, date)

Usage

```
$myArrayList.contains(object o)
```

- Returns true if object o is an element of the ArrayList. Otherwise, returns false.

```
$myArrayList.indexOf(object o)
```

- Returns the index position of the first occurrence of object o in the ArrayList. If object o is not in the ArrayList, -1 is returned.

```
$myArrayList.isEmpty()
```

- Returns true if the ArrayList contains no elements. Otherwise, returns false.

```
$myArrayList.lastIndexOf(object o)
```

- Returns the index position of the last occurrence of object o in the ArrayList. If object o is not in the ArrayList, -1 is returned.

```
$myArrayList.size()
```

- Returns the number of elements within the ArrayList.

Sorting Array Lists

ArrayList	$util.sort(arrayList a, string sortString)
ArrayList	$util.numericSort(arrayList a, string sortString)

Usage:

- **sortString**– The string defining sort orders. Note for numeric sorting, sortString should be set to the generic 'toString' argument, e.g. $util.numericSort($yourArray,"-toString")

ArrayLists comprised of any type of object (be it Clickability Platform content objects or standard object classes like strings and integers) – *except* custom objects created by $util.newObject – can be sorted by specifying the sort criteria in the form of a sort string.

Sort String

The sort string directs the manner in which objects in the ArrayList are sorted. It is a comma-separated list of elements of (or other expressions derived from) the objects being sorted, plus ascending/descending designators. As an example, the sort string "+title" would sort the items in the array based on their title, alphabetically. The sort string "-title" would sort the items based on the objects' titles, only in *reverse* alphabetic order. For numeric sorting, set the sort string to the generic 'toString' argument, e.g. $util.numericSort($yourArray,"-toString").

ArrayLists can also be sorted with multiple sorts, such as "+title,+summary" which would sort first based on objects' titles in alphabetical and then by their summaries. This is illustrated in the example below, in which the $cms.links tag (detailed in a later chapter of this guide) is used to retrieve a list of content items from the content management system, that are then sorted.

Multiple-Sort Example: Sorting an ArrayList of content objects

```
#foreach($item in $util.sort($cms.links, '-title,-summary'))
<p><b>$item.title</b><br/>
$item.summary</p>
#end
```

Advanced Sorting String

Advanced sorting strings can be used if it is anticipated that some of the sorted elements may be blank. The tag specifies a primary field to sort on, followed by a secondary element if the primary is empty for an individual item in the list.

For example, the following sort string sorts by "firstname" if there is one, but "fullname" if there isn't:

```
"+author.name.firstname|author.name.fullname".
```

Sorting on a Numeric Field

The special numericSort tag should be used if the field to be sorted on is a numeric value (integer or decimal) in a text or string field. A standard sort will sort numbers alphabetically, meaning that 11 will come before 2 since it starts with a one. The numeric sort utility will cast these fields as a number, and then compare them. All values that can not be casted successfully to a number (ie a string) will be placed at the end of the list in alphabetical order.

In the following example, listorder is a text entry field that represents the numeric ordering the content items should follow:

```
$util.numericSort($cms.links,"listorder")
```

Sorting an ArrayList of String Objects

If you need to sort an ArrayList of strings themselves (as opposed to sorting an ArrayList of Clickability Platform objects by a string element of the object), use "toString" as the sort string, like this:

```
#set($groups = ["Business", "Accounting", "Personnel"])
#foreach($group in $util.sort($groups, "toString"))
      $group<br/>
#end
```

Sorting an ArrayList of Integer Objects

If you need to sort an ArrayList of numeric values themselves (as opposed to sorting an ArrayList of Clickability Platform objects by a numeric-value element of the object), use empty quotes as the sort string, like this:

```
#set($ids = [2, 8, 3])
#foreach($id in $util.sort($ids, ""))
       <p>$id</p>
#end
```

Sorting an ArrayList by a Complex Object Property

You can use an object's methods as a sort string, provided the method can be expressed as a property (i.e., without the parentheses, like "categorization" for the "getCategorization()" method).

If, however, you want to sort on a value that can only be derived from the object by passing parameters into methods (e.g. categorization.getCategory(setID)), you must do so by first creating an intermediate array of OrderedPair objects that allow the desired value to be accessed as a property, then sorting on that.
An ordered pair is a collection of two objects (of any type) in a specific order.

In the below example, an ordered pair is used to associate a content item w/ its category value for a given category set (Set #646). By creating an ArrayList of these ordered pairs, we are then able to sort the OrderedPairs based on the category value (since getX() can be expressed simply as the property "x").

Example:

```
#set($orderedPairArraylist = $util.newArrayList)
#foreach($item in $cms.filteredLinks("Document","/documents",1,10))
  #set($o = $util.newOrderedPair())
  #set($foo = $o.setX($item.categorization.getCategory(646).name))
  #set($foo = $o.setY($item))
  #set($foo = $orderedPairArraylist.add($o))
#end

#foreach($o in $util.sort($orderedPairArraylist,"x"))
  #set($item = $o.getY())
  $item.title ($o.getX())<br/>
#end
```

ThreeTuples: Sorting an ArrayList by More than one Complex Object Property

If you need to sort an Array of objects by more than 1 complex property, you can use a ThreeTuple object, which forms a collection of 3 objects in a specific order.

In the below example, a ThreeTuple is used to facilitate sorting an array of Class content items first by a datecode, then a timecode, both of which are derived by complex logic not depicted here, and are then stored as properties of the ThreeTuple, along with the Class item itself. By creating an ArrayList of these ThreeTuple objects, we are then able to sort the ThreeTuples based on the complex values (which are simply access as “x”, “y”, and “z” using property notation), and extract the Class item back out as a simple property of the ThreeTuple.category value.

Example:

```
#set($timecodeDatecodeThreeTuples = $util.newArrayList)
#foreach($class in $resultsArray)
#set($threetuple = $util.threeTuple($timeCode,$dateCode,$class))
#set($foo = $timecodeDatecodeThreeTuples.add($threetuple))
#end
## Sort the threetuples first by the datecode argument (y), then the
timecode argument (x) derived from the class's conferenceTime
#foreach($threetuple in
$util.numericSort($timecodeDatecodeThreeTuples,"y,x"))
#set($class = $threetuple.getZ())
<li>$class.title</li>
#end
```

Unlike OrderedPairs, you cannot create an empty ThreeTuple; rather, you must declare all 3 properties of the ThreeTuple upon defining it.

ThreeTuple	$util.threeTuple(object x, object y, object z)
Given a ThreeTuple Object $myThreeTuple:	
ThreeTuple	$myThreeTuple.setX()
ThreeTuple	$myThreeTuple.setY()
ThreeTuple	$myThreeTuple.setZ()
ThreeTuple	$myThreeTuple.getX()
ThreeTuple	$myThreeTuple.getY()
ThreeTuple	$myThreeTuple.getZ()

HashMaps

Hashmaps are objects that contain a set of elements. However, unlike ArrayLists, which are indexed by a serial integer value, hashmap elements are indexed by user-created key objects. These key objects can be of any type, including integers, strings, etc.

Creating a HashMap

HashMap	$util.newHashMap

$util.newHashMap returns a new, empty hashmap object.

Adding, Retrieving and Removing Elements

```
#set($myHashMap = $util.newHashMap)

        $myHashMap.put(stringkey, object value)
        $myHashMap.get(stringkey)
        $myHashMap.remove(stringkey)
```

Usage

```
$myHashMap.put(object key, object value)
```

- This tag adds the object value to the hashmap with the object key used as the index key.

```
$myHashMap.get(object key)
```

- This tag retrieves the object from the hashmap that is indexed with the index value object key.

```
$myHashMap.remove(object key)
```

- This tag removes the object from the hashmap that is indexed with the index value object key.

Getting Information about the Object

Boolean	$myHashMap.containsKey(object key)
Boolean	$myHashMap.containsValue(object value)
Boolean	$myHashMap.isEmpty
Integer	$myHashMap.size
ArrayList	$myHashMap.keySet

Usage

```
$myHashMap.containsKey(object key)
```

- Returns true if the hashmap contains an object indexed with the specified index key.

```
$myHashMap.containsValue(object value)
```

- Returns true if the hashmap contains the object specified index key.

```
$myHashMap.isEmpty
```

- Returns true if the HashMap contains no elements. Otherwise, returns false.

```
$myHashMap.size
```

- Returns the number of elements within the hashmap.

```
$myHashMap.keySet
```

- Returns an ArrayList of the keys used to index elements of the hashmap.

User Defined Objects

User defined objects are data structures that are constructed to contain any number of variables representing objects of any type. Generally custom objects are used as a way of associating various properties and organizing concepts so code can be clean and easily supportable.

Here is an example of the type of structures that can be created:

Variable	Type	Example Value
Age	Integer	29
Favorite Color	String	"red"
Bio	Content Item	
Favorite Foods	ArrayList	["pizza","sushi","French fries"]
Birthdate	Date	July 28, 1973 20:20:00 EST

Creating an Object

Object	$util.newObject

$util.newObject returns a new, empty user object.

Adding Elements

```
$myObject.addString(string var, string str)
$myObject.addDate(string var, date date1)
$myObject.addInt(string var, int i)
$myObject.addBoolean(string var, boolean b)
$myObject.addArrayList(string var, array-list a)
$myObject.add(string var, object o)
```

There are type-specific add methods to add a variable to the object. If there is a type conflict between the type specified and the object provided, the variable will not be added to the user object. The generic $myObject.add method does not perform any type checking and may be used with any object type.

Usage

```
$myObject.addString(string var, string str)
```

- Adds a string variable named var to the object with the value str.

```
$myObject.addDate(string var, date date1)
```

- Adds a date variable named var to the object with the value date1.

```
$myObject.addInt(string var, int i)
```

- Adds an integer variable named var to the object with the value i.

```
$myObject.addBoolean(string var, boolean b)
```

- Adds a boolean variable named var to the object with the value b (either true or false)

```
$myObject.addArrayList(string var, array-list a)
```

- Adds an ArrayList variable named var to the object with the value a.

```
$myObject.add(string var, object o)
```

- Adds any object to the user object as a variable named var.

Retrieving Elements

String	$myObject.getString(string var)
Date	$myObject.getDate(string var)
Boolean	$myObject.getBoolean(string var)
Integer	$myObject.getBoolean(string var)
Integer	$myObject.getBoolean(string var)
Integer	$myObject.getBoolean(string var)
Integer	$myObject.getBoolean(string var)
Integer	$myObject.getInt(string var)
ArrayList	$myObject.getArrayList(string var)
Object	$myObject.get(string var)

There are type-specific “get” methods for all the standard basic object types. These methods perform type checking to ensure that the correct type is returned. There is also a generic “get” method that can be used to retrieve any type object without any type checking.

Usage

```
$myObject.getString(string var)
```

- Retrieves the string variable specified by the name var.

```
$myObject.getDate(string var)
```

- Retrieves the date variable specified by the name var.

```
$myObject.getInt(string var)
```

- Retrieves the integer variable specified by the name var.

```
$myObject.getBoolean(string var)
```

- Retrieves the boolean variable specified by the name var.

```
$myObject.getArrayList(string var)
```

- Retrieves the ArrayList variable specified by the name var.

```
$myObject.get(string var)
```

- Retrieves the variable specified by the name var.

Note: properties are also available by using regular dot notation instead of using .get() methods.

Examples:

This example outputs a list of speakers:

```
#set($speaker1 = $util.newObject)
$speaker1.add("name", "John Doe")
$speaker1.add("expertise", ["Web Analytics"])
$speaker1.add("isBookSigning", false)

#set($speaker2 = $util.newObject)
$speaker2.add("name", "Jane Smith")
$speaker2.add("expertise", ["SEO", "SEM"])
$speaker2.add("isBookSigning", true)

#foreach($person in [$speaker1, $speaker2])
    <div>
        <p>Name: $person.name</p>
        <p>Expertise:</p>
        <ul>
            #foreach($topic in $person.expertise)
                <li>$topic</li>
            #end
        </ul>
        #if($person.isBookSigning)
            <p><em>Signing books afterwards.</em></p>
        #end
    </div>
#end
```

This example outputs a list of items grouped per year:

```
## Get Items
#set($reports = $util.newArrayList)
#foreach($item in $util.sort($cms.filteredLinks("Annual Report", "/annual-
report", 1, 500), "-startDate"))
    #set($foo = $reports.add($item))
#end

## Separate By Year
#set($currentYear = "0")
#set($fiscalYears = $util.newArrayList)
#set($tempArray = $util.newArrayList)
#set($tempObj = $util.newObject)
#foreach($item in $reports)
    #if(!$util.isNull($item.startDate))
        #if("$util.date('yyyy', $item.startDate)" != $currentYear)
            #if($tempArray.size() > 0)
                #set($foo = $tempObj.add("items", $tempArray))
                #set($foo = $tempObj.add("year", $currentYear))
                #set($FOO = $fiscalYears.add($tempObj))
            #end
            #set($tempObj = $util.newObject)
            #set($tempArray = $util.newArrayList)
            #set($currentYear = "$util.date('yyyy', $item.startDate)")
        #end
        #set($foo = $tempArray.add($item))
```

```
    #end
#end
#if($tempArray.size() > 0)
    #set($foo = $tempObj.add("items", $tempArray))
    #set($foo = $tempObj.add("year", $currentYear))
    #set($FOO = $fiscalYears.add($tempObj))
#end

## Display Items per Year
#foreach($obj in $fiscalYears)
    <p class="header">
       Fiscal Year $!obj.year
    </p>
    <ul>
       #foreach($item in $obj.items)
           <li>
              <a href="$!item.externalURL">$!item.title</a>
           </li>
       #end
    </ul>
#end
```

Getting Information about the Object

ArrayList	$myObject.variables
ArrayList	*$myObject.values*
Boolean	$myObject.isEmpty
Integer	$myObject.size

Usage

```
$myObject.variables
```

- Returns an ArrayList of the variable names in the user object.

```
$myObject.values
```

- Returns an ArrayList of the variable values in the user object.

```
$myObject.isEmpty
```

- Returns true if the user object contains no elements. Otherwise, returns false.

```
$myObject.size
```

- Returns the number of variables within the user object.

Using Sub-Templates and Include Files

Using Sub-Templates

String	$cms.template(string templateName)
String	$cms.template(int templateId)
String	$cms.cachedTemplate(string templateName, string key [, int TTL])

Usage:

- **templateName** – The name of the template to be included.
- **templateId** – The system ID of the template to be included.
- **key** – The key is a unique string variable that the template is cached on. For instance, if the results of the template vary by content ID, the content ID as a string should be passed in as the key.
- **TTL** – A TTL in seconds; if excluded the default TTL is 60 seconds.

The $cms.template tag inserts the specified template into the current template. The calling template and the included template(s) are then processed as a whole, such that all Clickability Platform template language tags are processed. Circular references within templates are not supported (ie. a template can not call itself in a $cms.template tag).

Examples:

```
$cms.template("LeftNav")
$cms.template("Header")
$cms.template(491)
```

The $cms.cachedTemplate tag is intended for sub-templates that are placed on every webpage (such as a header or footer) and make expensive calls. In order to increase efficiency, the results of those sub-templates can intentionally be cached so the expensive calls aren't requested repetitively. The $cms.cachedTemplate tag obeys production caching rules.

Example:

```
$cms.cachedTemplate("Related Links", "$content.id", 600)
```

Accessing Variables in Sub-Templates

When a template is called into another template via the $cms.template tag, the included template is referred to as a sub-template. In working with sub-templates, variables are accessible across templates according to the following rules:

A variable defined in a template may be used in a sub template provided that the variable definition is before the sub-template tag call.

In the following example:

```
#set($name = "Edgar")
$cms.template("Header")
#set($address = "Downtown")
```

the variable $name is available in the “Header” sub-template, but the variable $address is not.
Similarly, a variable defined in a sub template may be used in the parent template after the sub-template call.

For example, if the “Footer” sub-template sets the variable $color to the value Green in the following example:

```
#set($color = "Red")
$color
$cms.template("Footer")     ##sets $color to Green
$color
```

the resulting HTML will be:

- Red
- Green

because the first reference occurs before the variable is defined in the sub-template.

Retrieving Template Data

Integer	$cms.templateData(int id).id
Integer	$cms.templateData(string name).id
String	$cms.templateData(int id).text
String	$cms.templateData(string name).text
Integer	$cms.templateData(int id).version
Integer	$cms.templateData(string name).version
Date	$cms.templateData(int id).createDate
Date	$cms.templateData(string name).createDate
Date	$cms.templateData(int id).lastModified
Date	$cms.templateData(string na).lastModified

Usage:

- **name** – The name of a template
- **id** – The system ID of a template

The $cms.templateData tag retrieves information about a template, without processing the tags within the template.

Usage

```
$cms.templateData("topNav").id
```

- Retrieves the numeric ID value of the template named “topNav”.

```
$cms.templateData(256).name
```

- Retrieves the name of the template with numeric ID value of 256.

```
$cms.templateData("topNav").text
```

- Retrieves the contents of the template named “topNav” as code, i.e., *without* processing the Clickability Platform template language tags within the template.

```
$cms.templateData(256).version
```

- Retrieves the version of the template utilized in the current publisher environment. If this tag is being processed in the Development publishing environment, the value retrieved will be the version of the template currently assigned to Development.

```
$cms.templateData("topNav").createDate
```

- Returns the date that the topName template was created.

```
$cms.templateData("topNav").lastModified
```

- Returns the date that the topName template was last modified.

Include Files

String	$cms.include(string file-name).text
String	*$cms.include(string file-name).url*
String	$cms.include(int includeID).text
String	$cms.include(int includeID).url

Usage:

- **file-name** – The file name of the include file.
- **includeID** – The system ID of the include file.
- **attribute-name** – The attribute to be referenced.

Allows inclusion of the specified include file in the current template as static text or as a linked url. Template processing WILL NOT occur on the included file (unless the include text is processed by a $cms.translate tag in the calling template, as described in the “Formatting Text” chapter).

Include File Examples:

```
$cms.include("mainStyleSheet").text
$cms.include("topJavascript.js").url
$cms.include(318).url
```

The *url* attribute can be used in templates as follows:

```
<link rel="StyleSheet" href="$cms.include('mainStyleSheet').url"
type="text/css">
<script language="JavaScript1.1" src="$cms.include('topJavascript.js').url"
type="text/javascript"></script>
```

The *text* attribute can be used in templates as follows:

```
<style type="text/css">
$cms.include("mainStyleSheet").text
</style>
<script language="JavaScript1.1">   $cms.include("topJavascript.js").text
</script>
```

Retrieving Include Data

Integer	$cms.includeData(int id).id
Integer	$cms.includeData(string *name*).id
String	$cms.includeData(int *id*).text
String	$cms.includeData(string *name*).text
Integer	$cms.includeData(int *id*).version
Integer	$cms.includeData(string *name*).version
Date	$cms.includeData(int *id*).createDate
Date	$cms.includeData(string *name*).createDate
Date	$cms.includeData(int *id*).lastModified
Date	$cms.includeData(string *na*).lastModified

Usage:

- **name** – The name of an include
- **id** – The system ID of an include

Content Tags

Content Item Elements

ContentObject	$cms.content
ContentObject	*$cms.getContent(int contentID, String websiteSection-name|website-path)*
ContentObject	`$cms.getContent(int contentID, int domainID, String websiteSection-name|website-path)`
ContentObject	$cms.getContent(int contentID)*

Given a Content object $item:

Object	$item.attribute-name
Object	$item.element(String attribute-name)
ArrayList of Content IDs	`$item.linkedContent(String linkedContentTag [,contentTypeNameOrID])`
Object	$item.isDefined(String attribute-name)
Boolean	$item.isHTMLField(String attribute-name
Boolean	$item.isTextField(String attribute-name)
Integer	$item.version

Usage:

- **contentid** – The specific contentID of the content item to be used. When omitted, the publisher defaults to the content item currently being published. (see warning below)
- **website-section-name** – The name of the website section where you are referring to the content by ID. Defaults to the current website section being published
- **website-path** – The path (relative from domain) to the websiteSection where you are referring to the content by ID. Defaults to the current websiteSection being published.
- **domainID** – The id of the domain to which the indicated website-section-name or website-path belongs (if not the current one)
- **attribute-name** – The name of the content attribute to be output (derived from the content-type definitions) or a system attribute or constructed attribute.
- **linkedContentTag** – The tag name of the linked content browser containing content items you wish to retrieve.
- **contentTypeNameOrID** – The content type (by Name or ID) of the items you wish to retrieve from the linked content browser item.

System Attributes:

- id
- modifiedDate
- createDate
- contentLiveDate
- contentArchiveDate
- contentDeadDate

- typeName
- typeID
- typeVersion
- URLName (works only if custom URL resource names enabled)

Constructed Attributes:

- url
- contentTargets

* WARNING:

When using $cms.getContent(contentId) without specifying a website section for a URL, the publisher will first check for a target in the website section being published for the URL attribute. If there is no target in that website section, $cms.getContent(contentId) will use the priority 1 target for that content item as the URL. If the content item has no targets, a null URL will be returned.

URL of a Content Item

A single piece of content can be targeted to multiple website sections, giving that content page a different URL in each of those sections. If you successfully retrieved the item from the current section via $cms.getContent(ID), or from a specific section via $cms.getContent(ID, sectionPathOrName) it will use that section's path to construct the url.

Note: Previously, to access content from a template when a user is unsure where the content is targeted, users were instructed to use the Velocity function, $cms.content(id, false), then use the .contentTargets logic to find the appropriately targeted version of the Content Item. Instead, the function $cms.getContent has been created to better access content from a template in a variety of scenarios. The previous uses of $cms.content have been deprecated.

Content Targets

Given a Content object $item

ArrayList of ContentTarget Objects	$item.contentTargets
Object	Given a ContentTarget object $target: $target.attribute-name

Attributes:

- destID (or destinationID)
- path (or destPath)
- domainID
- domain
- url
- priority – the order of this target on the Targets tab
- isLive (or just live)
- isArchive (or just archive)
- isDead (or just dead)
- createDate

- startDate
- archiveDate
- deadDate

Simple Content Fields

In the simplest form, the $cms.content tag is used to display a specific attribute of the content item being published. For example, your content type may be defined to have the following attributes: title, summary, body. The following tags would retrieve these fields:

```
$cms.content.title
$cms.content.body
$cms.content.summary
OR $cms.content.element("title")
```

System information may also be retrieved for the content item. The following tags will return the content item's "id", its "last modified" date, the date the item was created, and the date the item went live, respectively.

```
$cms.content.id
$cms.content.modifiedDate
$cms.content.createDate
$cms.content.contentLiveDate
```

Similarly, information regarding the "Content Type" of the content item is retrievable through the following tags, which retrieve the Content Type name, id number, and version number for the content item, respectively.

```
$cms.content.typeName
$cms.content.typeID
$cms.content.typeVersion
```

Finally, the URL of the content item being published can be retrieved with the following tag. This URL will be constructed based on content targeting information and the website section being published.

```
$cms.content.url
```

Multi-value Fields

To retrieve multi-value fields from the content item, it is necessary to combine the $cms.content tag with a #foreach loop as the data is returned as an ArrayList.

For example,

```
<ul>
       #foreach( $person in $cms.content.authors )
       <li>$person</li>
       #end
</ul>
```

would output:

- Alice Author
- Warren Writer
- Ricky Reporter

This #foreach loop causes the “cms.content.authors” list to be looped over for all of the authors in the list. Each time through the loop, the value from “cms.content.authors” is placed into the $person variable.

Complex Multi-value Fields (e.g. Complex Fields)

As an example of a complex multi-value field, consider a set of baseball games box scores, each consisting of a home team, a home team score, an away team, and an away team score. To display these fields, it is a #foreach loop is utilized which returns the complex box score fields, which can then be referenced within the loop.

```
#foreach( $box in $cms.content.boxScore )
	$box.homeTeam <b>$box.homeScore </b><br/>
	$box.awayTeam <b>$box.awayScore </b><br/>
	<br/><br/>
#end
```

would output:

- RedSox 12
- Yankees 3

- San Francisco 9
- Dodgers 2

The template engine provides an easy way to get the loop counter so that you can do something like the following:

```
#foreach($person in $cms.content.authors)
	$counter: $person<br/>
#end
```

would output:

- 1: Alice Author
- 2. Warren Writer
- 3: Ricky Reporter

Note: The object returned by a complex field element is not a true ArrayList object, so you cannot use the normal ArrayList methods like size() or isEmpty(). All you can do with it is iterate using #foreach.

Yes/No Fields

Yes/No fields return a Boolean value when retrieved in the templates. The value is either true or false and may be checked in a conditional statement as follows, where showHeading is a Yes/No field.

```
#if($cms.content.showHeading)
	$cms.content.showHeading
#end
```

Since these are Boolean values, when submitting values for Yes/No fields to the Clickability Platform (e.g. via Submission Queues) you must pass a value of “true” or “false”.

Item Browser Fields

An Item Browser content element returns an ArrayList* of content IDs, which can then be used to retrieve the actual content items as desired.

* **Note:** The ArrayList returned by an item browser element is not a *true* ArrayList object, so .add() and .remove() functions are not recommended.

Example: Assuming that subArticles is an Item Browser field:

```
#foreach($itemID in $cms.content.subArticles)
	#set($item = false)
	#set($item = $cms.getContent($itemID))
	#if($item)
		$item.title <br/>
	#end
#end
```

Linked Content Fields

Like an Item Browser, a Linked Content element returns an ArrayList* of content IDs, which can then be used to retrieve the actual content items as desired.

* **Note**: The object returned by a linked content element is not a *true* ArrayList object, so you cannot use the normal ArrayList methods like size() or isEmpty(). All you can do with it is iterate through the array using #foreach.

Example: Assuming that “Movie” and “Review” are content types that the linked content element tagged “moviesAndReviews” is configured to accept:

```
$cms.content.linkedContent("moviesAndReviews","Review")
```

would return a list of the IDs of all “Review” items assigned to the current content item’s “Movies and Reviews” linked content field, which you could then use to look up the corresponding Review content objects as follows:

```
#foreach($itemID in $cms.content.linkedContent("moviesAndReviews","Review"))
#set($item = false)
#set($item = $cms.getContent($itemID))
#if($item)
$item.title <br/>
#end
#end
```

Select List Fields

A Select List content element returns a string of the selected value. Note that if that string needs to be compared to another value, it must be compared using the $string.equals method (and not via ==). For example:

```
#if($string.equals($cms.content.color, "blue"))
	The color is Blue
#end
```

Date and Date/Time Fields

Date and Date/Time fields return Date Objects when retrieved in the templates.

Updates Fields

The "Updates" field is a special field type designed to allow for tracking publishable updates to a content item, as captured by 4 properties per update: an "Author" field, an auto-populated "Date/Time" timestamp, a "Subject" field, and a "Body" field. The time stamp is populated on save, and the Author field is populated by the current WCM user's name (though this can be configured to be overridden to accept manual changes). The Update field's "Body" property can be configured to use the HTML Editor if desired.

The syntax for accessing the collection of updates contained in an Updates field is:

```
$cms.content.[updateField].items
```

Where [updateField] is whatever tagname was assigned to the Updates field in the Content Type definition.

This will return an arrayList of updates (in 'oldest first' order) that can then be iterated using a #foreach loop like so:

```
#foreach($update in $cms.content.myUpdates.items)
        Update by $update.author on $update.updatedDate:<br/>
        $update.subject
        <p>$update.body</p>
#end
```

Media within Content Items

A content item's media placements (single media asset associations) and media arrays (groupings of media asset associations) can be accessed as MediaPlacement objects.

The tag for a traditionally-structured 'media placement' will return just that single MediaPlacement object, whereas the tag for a 'media array' will return an ArrayList of all the [ordered] MediaPlacements in the grouping.

Note: A MediaPlacement Object differs from the full MediaObject described in the "Media" chapter. Whereas a MediaPlacement represents a specific usage of a MediaObject within a particular content item, a MediaObject is the actual media repository asset from which the content item's media instances are derived.
To get the URL of a media placement, the corresponding MediaObject should be used.

MediaPlacement	$contentItem.[mediaPlacementTag]
ArrayList of MediaPlacements	*$contentItem.[mediaArrayTag]*
ArrayList of all single-asset MediaPlacements	$contentItem.assignedMedia
ArrayList of all single-asset MediaPlacements	$contentItem.assignedMedia(String mediaType)

Given a MediaPlacement object $placement:

Attribute Value	$placement.attribute-name

Usage:

- $contentItem – Is a content item object, obtained through a $cms.content tag or other means.
- [mediaPlacementTag] – the tagname of the media placement

- [mediaArrayTag] – the tagname of the media array
- attribute-name – The tagname of the media attribute to be output
- mediaType – A media type: images, audio, video, documents, binaries

Default Attributes (for images):

- title
- name (returns the tagname)
- caption
- height
- width
- altText
- author
- filename
- extension
- metadata
- id
- target
- targetURL

Constructed Attributes (for images):

- $util.tag(MediaPlacement media)
- $util.resizeImageTag(MediaPlacement media)
- $util.linkedTag(MediaPlacement media)

Media Placement Tag Examples:

```
$cms.content.topImage.caption
$cms.content.leftimage.altText
$cms.content.video1.title
$cms.content.topImage.caption
```

The URL should be retrieved from the corresponding MediaObject, using the

```
$cms.media tag:
$cms.media($cms.content.topImage.id).url
```

or

```
$cms.media($cms.content.topImage, "images").url
```

would result in:

- http://media.website.com/images/filename.gif

This URL is based on the media domain configured in the Clickability Platform for the website being published.
As a shortcut, the complete HTML image tag can be generated for image media assets, as explained in the “Media” chapter.

```
$util.tag($cms.content.TopImage)
```

would result in:

```
<img src="http://images.website.com/images/filename.gif" height="100"
width="100" alt="This is the alttext" border=0 />
```

Retrieving All Assigned Media Items

There are tags to retrieve all of the *single-asset* (i.e. media placement) media assigned to a content item, either by specific media type, or all types together.

```
$cms.content.assignedMedia
$cms.content.assignedMedia("Images")
$cms.content.assignedMedia("Video")
$cms.content.assignedMedia("Audio")
$cms.content.assignedMedia("Documents")
$cms.content.assignedMedia("Binaries")
```

Note: the .assignedMedia() tags do NOT return media from media *arrays*, only single media placements.

Calendar Schedules on Content

ArrayList of Schedule Objects	$contentItem.calendarSchedules

Given a Schedule object $schedule:

Object	$schedule.attribute-name

Attributes:

- **contentID** – the ID of the content item to which the Schedule applies
- **scheduleType** – possible values: ONCE, DAILY, WEEKLY, MONTHLY_DAY_OF_WEEK, MONTHLY_DAY_OF_MONTH
- **startDate** – returned as a Date String in format yyyy-MM-dd
- **endDate** – returned as a Date String in format yyyy-MM-dd
- **startDay** – the day of the month as an Int
- **endDay** – the day of the month as an Int
- **startMonth** – the month as an Int (where 1 = January)
- **endMonth** – the month as an Int (where 1 = January)
- **startYear –** the year as an Int
- **endYear** – the year as an Int
- **startHour** – the hour as an Int
- **endHour** – the hour as an Int
- **startMinute** – the minute as an Int
- **endMinute** – the minute as an Int
- **startTime** – the Time String (e.g. 22:55:00)
- **endTime** – the Time String (e.g. 22:55:00)
- **startTimeFormatted** – formatted time string (e.g. 10:55 PM)
- **endTimeFormatted** – formatted time string (e.g. 10:55 PM)
- **location** – the entered location for the Schedule
- **details** – the entered details for the Schedule
- **description** – a system-output description for the Schedule

- **oneTime** – returns true for a one-time Schedule
- **daily** – returns true for a daily Schedule
- **weekly** – returns true for a weekly Schedule
- **monthly** – returns true for [either type of] monthly Schedule
- **monthlyByDayOfWeek** – returns true for a monthly-by-day-of-week Schedule
- **monthlyByDayOfMonth** – returns true for a monthly-by-day-of-month Schedule
- **weekly**[Sunday - Saturday] –returns true for a weekly Schedule that includes [the indicated day of week]
- **weeklyDays** – returns comma-separated list of days that the weekly Schedule is scheduled for (e.g. Monday, Wednesday)
- **monthlyDayOfWeek** – returns the day of week (String) for monthly-by-day-of-week Schedules
- **monthlyOrdinal** – returns the ordinal for monthly-by-day-of-week Schedules. Possible values: 'first','second','third','fourth','last'
- **monthlyDayOfMonth** – returns the day of month (Int) for monthly-by-day-of-month Schedules

Content items can make use of Calendar Schedules via the Calendar tab, (which is enabled on the Content Type settings by checking "Enable events").

For some content item $cms.content which has schedules assigned,

```
$cms.content.calendarSchedules
```

will return a list of Schedule objects. A schedule can be entered as one-time, or entered as a recurring type that happens daily, weekly, monthly, etc.

Given a Schedule object $schedule:

ArrayList of Occurrence Objects	$schedule.getOccurrences([DateString fromDate, DateString toDate])

Usage:

- **fromDate** – a 10-character string representing te start of the range for which you want to return Occurrences, in the format 'yyyy-MM-dd'
- **toDate** – a 10-character string representing the end of the range for which you want to return Occurrences, in the format 'yyyy-MM-dd

Given an Occurrence object $occurrence

Object	$occurrence.attribute-name

Attributes:

- **schedule** – the Schedule Object for which this event is an occurrence
- **date** – the Date String representing the date of this occurrence *

***Note**: This is a String because of potential time zone conflicts in the standard java Date class; use the $util context date parsing function for conversion from a String to a Date object

- **day** – the day (Int) of this occurrence
- **month** – the month (Int) of this occurrence
- **year** – the year (Int) of this occurrence

Each Schedule can be expanded into a list of [single-day] Occurrence objects for that schedule.

Examples:

To show all occurrences of all schedules for a content item $cms.content:

```
#foreach($schedule in $cms.content.calendarSchedules)
       #foreach($occ in $schedule.occurrences)
<li>$occ.date - from $occ.schedule.startTimeFormatted to
$occ.schedule.endTimeFormatted</li>
#end
#end
```

To show a schedule of events:

```
#foreach($item in $cms.filteredLinks("Event",1,5))

        #foreach($schedule in $item.calendarSchedules)

            #if($schedule.daily)
                #set($theSchedule = "Daily Schedule - From
$schedule.startMonth/$schedule.startDay -
$schedule.endMonth/$schedule.endDay")

            #elseif($schedule.weekly)
                #set($theSchedule = "Weekly Schedule - $schedule.weeklyDays
- From $schedule.startMonth/$schedule.startDay -
$schedule.endMonth/$schedule.endDay")

            #else
                #set($theSchedule = "Scheduled only One Time,
$schedule.endMonth/$schedule.endDay")
            #end

        #end

        <p><strong>$!theSchedule</p>
            $item.body
        <p><a href="$item.url">Click here for Details</a></p>

#end
```

Categorization of Content Items

To get a Categorization object:

Categorization Object	$contentItem.categorization

Given a Categorization object $catgzn

ArrayList of CategorySet Objects	$catgzn.sets
ArrayList of Ints	$catgzn.setIDs
Boolean	$catgzn.allowsMultiple(int setID or String setName)
ArrayList of Category Objects	$catgzn.getCategories(int setID or String setName)
Category Object	$catgzn.getCategory(int setID or String setName)

Usage:

- **$contentItem** – Is a content item object, obtained through a $cms.content tag or other means.
- **setID** – The id of the CategorySet of interest
- **setName** – The name of the CategorySet of interest

You can derive the categorization settings of a specific content item by calling getCategorization() on that item.

The CategorizationObject that is returned can then be used to derive the relevant CategorySet and Category objects, from which additional categorization details (such as name, level, parent) can be obtained using the methods described in the “Categories” chapter.

Accessing Content Language Information

The following is available if a given content object has a Language assigned.

Given a Content object $content:	
Boolean	$content.languageInfo
String	$content.languageInfo.tag
String	$content.languageInfo.name
String	$content.languageInfo.font
Integer	$content.languageInfo.fontSize
Boolean	$content.languageInfo.rtl
String	$content.languageInfo.styles
String	$content.languageInfo.dir

```
$content.languageInfo.tag
```

- Returns the language tag for the language, e.g. 'en' for English or 'pt-BR' for Brazilian portugese.

```
$content.languageInfo.name
```

- Returns the name/description of the language.

```
$content.languageInfo.font
```

- Returns the font for the language (will return an empty string "" if none is available).

```
$content.languageInfo.fontSize
```

- Returns the font-size in pixels (if assigned, otherwise will return 0).

```
$content.languageInfo.rtl
```

- Determines if the language is Right-to-Left (returns true if this is a Right-to-left language such as Arabic, or Hebrew; false otherwise).

```
$content.languageInfo.ltr
```

- Returns true, if this is a Left-to-right language such as English; returns false otherwise.

```
$content.languageInfo.styles
```

- Returns the relevant part of a styles attribute or CSS, to be used in an HTML tag; this is based on the font and fontSize values. A value for styles may be: “font-family: arial; font-size: 13px;” - the value will be "", if neither a font or fontSize is assigned.

```
$content.languageInfo.dir
```

- Returns the direction of text input on the Content Item (either 'rtl' or 'ltr'. These are the values that should be specified in HTML dir attribue), e.g: Some text .

Example

```
#if($content.languageInfo)
## Content has Language assigned
#else
## Content does not have Language assigned
#end
```

Accessing Deleted Content

It is possible to get limited information about content items that were deleted from the Clickability Platform or removed from the website.

To get items deleted within the last n days:

ArrayList of DeletedContent Objects	*$cms.getDeletedContentItems(int n)*

To get itemsDeletedContent objects deleted within a specific date range:

ArrayList of DeletedContent Objects	$cms.getDeletedContentItems(Date startDate, Date endDate)

Given a Deleted Content object $deletedContent:

Object	$deletedContent.attribute-name

Usage:

- **n**– the number of days back from now for which deleted content should be shown
- **startDate** – the start of the date range for which deleted content should be shown
- **endDate** – the end of the date range for which deleted content should be shown
- **attribute-name** is one of the DeletedContent Object attributes below:

Attributes:

- **title** – the Title of the content item
- **contentID**– the ID of the content item
- **date** – the date and time of the last action on this item

Comments

Basic Comments

To get the number of approved comments for a content object:

Int	*$cms.content.commentCount*

To get the comments for a content object:

ArrayList of Comment Objects	$feedback.comments
ArrayList of Comment Objects	$feedback.getComments(int contentItemID)
Comment Object	$feedback.getComment(int commentID)

To get the defined maxChars and commentPeriod values:

Int	$feedback.commentLength*
Int	$feedback.commentPeriod*
	*works only on detail pages, where the content type can automatically be ascertained

Given a Comment object $myComment:

Object	$myComment.attribute-name

Usage:

- **contentItemID** – the ID of the content item with the desired comments (only required if not current item)
- **commentID** -- the ID of the desired comment item
- **attribute-name** – one of the Comment object attributes below:
- **Attributes** (for Comment objects):
- **title** – the title of the comment, if specified
- **comment** – the comment body
- **name** – the Name submitted with the comment, if specified
- **website** – the Website submitted with the comment, if specified
- **IPAddress** – IP address from which the comment was submitted
- **contentTitle** – the Title of the content item with the comment
- **contentID** – The content ID of the content item with the comment
- **commentDate** – the date the comment was submitted
- **modifiedDate** – the date the comment was modified (in the Clickability Platform)
- **status** – the status of the comment (FLAGGED, APPROVED, NOTAPPROVED, AUTHOR)
- **numOfNaughtyWords** – the number of words matched in the flagged term filter
- **ID or commentID** – the comment ID

Retrieving Comments for Display

You can get and display the list of all approved comments for the current (or a specified) content item by using:

```
$feedback.comments
```

to get the comments for the current content object, or

```
$feedback.getComments(int contentItemID)
```

to get the comments for a content item that is not the current content item.

Note: These methods return you ONLY the Approved comments, so they can all be safely displayed on your page. Comments that are flagged or not approved will not be included in the arraylist returned by these tags.

Example to list out all the comments for the current Article page:

```
#foreach($comment in $feedback.comments)
       <div>Posted by $comment.name</div>
       <div>$comment.comment</div>
#end
```

If you are using traditional form submission to collect the comments, the above Velocity code would be all you need to display the complete list.

However, if you use AJAX to post new comments back to the page without reloading, you will need additional javascript to show the just-posted comments. The javascript vars for that AJAX code are noted in the "Coding the Comment Form" section below.

Configuring the Comment Form

You can retrieve the content-type-level configuration settings for the particular type of comment by using the following methods:

```
$feedback.commentLength
```

- which returns the number of allowed characters that were indicated, and

```
$feedback.commentPeriod
```

- which returns the number of days indicated as the allowable comment period.

These values would be used, for instance, in constructing the form validation or deciding whether or not to even show the comment form, respectively.

Enabling a Content Type for Commenting

Before comments can be accepted for a particular type of content, that content type must be configured in the Clickability Platform to allow commenting via the following fields on the "edit settings" link in the Admin > Content and Media > Content Types screen:

- **Yes, allow Visitor Feedback** – check this to turn on commenting
- **Flag forbidden terms** – check this to automatically flag comments that contain any words listed in the Flagged Term Manager.

- **Detect SPAM comments** – check this to enable bot checking on comments.
- **Yes, check email for flagged terms** – check this to check the email address for words listed in the Flagged Term Manager
- **Feedback Approval** – Select Automatic to let approved comments automatically publish on the site. Select Manual to require manual approval before publishing to site.
- **Allow Feedback For** – enter a [optional] number of days that the templates can reference to know how long to keep the comment form available for.
- **Comment Length** – enter a [optional] character limit that can be used to restrict the comment body.
- **Yes, allow Visitor Ratings** – check this to enable rating of content items with the specified Rating Type.
- **Prevent multiple ratings per computer via cookie check** – check this to limit ratings to one per customer per content item.
- **Yes, allow Visitor ranking of comment items** – check this to allow visitors to rank comments and specify the down-ranking threshold.
- **Send an email notification to the comment reviewer when a new comment is submitted** – check this to send an email when comments are submitted. This feature is only available if the Feedback Approval method is set to Manual.
- **Send an email notification to the comment submitter when a new comment is approved** – check this to send an email to the comment submitter when the comments are approved. This feature is only available if the Feedback Approval method is set to manual and the comment submitter email address is provided

Coding the Comment Form

To collect user comments via a form on your site, you will need to code the form with specific field names and a specific onsubmit function. You can choose whether it will submit via a standard method that then reloads the page, or whether it should submit using AJAX to dynamically update the page without reloading. Where these methods necessitate differences in the code, it will be noted. Everything else is common to both approaches.

Macro

The javascript for handling the bot detection, AJAX calls, and non-AJAX redirect option has been made available in a special macro, that must be included in your page *above* your comment form (but below your response function if you're using the AJAX method). The Velocity tag to include the macro is simply:

```
#commentCode()
```

OR – if your package includes the Social Media Toolkit –

```
#socialMediaCode()
```

Form Tag

The <form> element does not need to contain an action attribute; rather, its onsubmit must call one of the 2 method variants defined by the macro:

```
<form onsubmit="postForm(this)">
```

for regular post OR

```
<form onsubmit="postForm(this, true, handlerFunction)">
```

for AJAX post, where handlerFunction is the function that will execute the update to the page via the servlet information returned.

Form Fields

To properly map the collected data to the contet item contents, the form fields must use these designated name values (with the only required field being “field”):

- **name**
- **title**
- **email**
- **website**
- **field*** – this should be the main comment field. *Without this it will fail.
- **comment** – this should be a hidden field used for bot-detection.
  Note: not recommended for use with ratings.
- **path** – a hidden field used with non-AJAX comment forms to tell it where to go upon reload. This would also be used (in both AJAX and non-AJAX forms) if an error were to occur in the form processing. If not provided, path value will default to current page.
- **parentID** – if this field is in the form and the value designates the commentID of a valid and existing comment, a new, threaded comment will be created as a response to the comment specified. If the ‘parentID’ field is not in the form, or the value does not designate a valid comment, the new comment will be created as a basic top-level comment.

Post-back Javascript Function (for AJAX method only)

When using the AJAX submission utilities via #commentCode, #socialMediaCode and #socialMediaCodeNoLibrary macros, the new comment data will be populated to a wrapper javascript object: Clickability.Comment

The following fields may be set in the new object. Only the relevant fields will be set, if your comment form does not include a "name" field, for instance, name will remain null.

Variable	Description
Clickability.Comment.name	The given name of the comment submitter
Clickability.Comment.title	The title given to the comment
Clickability.Comment.email	The email address of the comment submitter
Clickability.Comment.website	The website address submitted by the commenter
Clickability.Comment.comment	The comment text
Clickability.Comment.flagged	A boolean specifying whether the comment has been flagged due to offensive content, or bot detection
Clickability.Comment.isAuthor	A boolean specifying whether the comment is an author comment
Clickability.Comment.status	possible status values: APPROVED, NOTAPPROVED, FLAGGED, AUTHOR, BOT, REJECTED
Clickability.Comment.error	The 'error' variable should be populated if an error occurred (i.e. if the form makes it to the returnFunction but nothing posts, an alert(error) can be done to see what is going wrong)
Clickability.Comment.id	The commentID of the newly added comment. Can be used as a parentID in future threaded responses to this comment

Note: Only the relevant variables will be set; if your comment form does not include a “name” field, for instance, name will remain null. The ‘error’ variable should be populated if an error occurred (i.e. if the form makes it to the returnFunction but nothing posts, an alert(error) can be done to see what is going wrong).

If using AJAX, these JS variables can be used to determine if the comment was APPROVED or not and

dynamically display the comment on the page. This is done via the function defined as the 'handlerFunction' argument in the postForm() call.

Comment Form Example

Note: portions highlighted in grey are used only for the AJAX method of posting back the submitted comment to the page without reloading. For non-AJAX method, those portions can be omitted.

```
<html>
<head>
<title>Comment Form</title>

<script type="text/javascript">
  function displayNewComment() {
    // Assumes "msg" element and "commentSection" elements in HTML

    var msgP = document.getElementById("msg");
    msgP.innerHTML = "Thank you! Your comment has been added below.";

    if(comment != "") {
      if(!flagged) {
var commentSection = document.getElementById("commentSection");
        if(name == "") {
          name = "Anonymous";
        }
        var nameDiv = document.createElement("div");
        nameDiv.setAttribute("class", "postedBy");
        nameDiv.innerHTML = "Posted by " + name;
        commentSection.appendChild(nameDiv);
        var commentDiv = document.createElement("div");
        commentDiv.innerHTML = comment;
        commentSection.appendChild(commentDiv);
      } else {
        msgP.innerHTML = "Comment has been sent for approval";
      }
    }
  }
</script>
</head>

<body>

  #commentCode()
<p id="msg"></p>
<form method="POST" onsubmit="return postForm(this, true,
displayNewComment);">
<div id="title">Post a comment!</div>
<div class="row">Name: <input name="name" type="text" /></div>
<div class="row">Comment: <textarea cols="20" rows="4"
name="field"></textarea></div>
<input type="submit" value="Submit">
</form>

<div id="commentSection">
  #foreach($comment in $util.sort($feedback.comments, "-ID"))
<div class="postedBy">#if($comment.name ==
"")Anonymous#else$comment.name#end said:</div>
<div>$comment.comment</div>
  #end
</div>
```

```
</body>
</html>
```

Note: The above examples all assume comments are being submitted from the content item's Detail page - i.e. that content item's own unique url.

If the comment form for a content item exists on a page other than its detail page, a "cid" JavaScript variable will need to be set after the macro, and before the <form>, like this:

For #commentCode():

```
<script>Clickability.Comment.Private.cid = $cms.content.id;</script>
```

For #socialMediaCode():

```
<script>Clickability.SMTK.cid = $cms.content.id;</script>
```

Writing a Comment Notification Template

Content Types that have comments enabled can be set up to send notification emails to a comment reviewer when a new comment is submitted, and/or to the comment submitter when their comment is approved (for comments set up with Feedback Type = Manual only).

The notification template is custom configured to contain the information that you specify. $util methods and all Comment Object properties are available.

Comment Notification Template Example

The following is an example of a comment notification template.

```
A comment has been submitted on content item ID: $comment.contentID <br/>
With the title: $comment.contentTitle<br/>
The comment ID is: $comment.id<br/>
The comment Title is: $comment.title<br/>
The comment was submitted by: $comment.name<br/>
The comment was submitted by IP Address: $comment.IPAddress<br/>
The comment date is: $comment.commentDate<br/>
The comment status is: $comment.status<br/>
There are $comment.numOfNaughtyWordsLanguage Filter Item Matches.<br/>
```

Ratings Comments

Setting up the Ratings Form

These are fields that can be added to the comment form so rating information can be submitted with a comment. Even though this goes in the system as a comment, it can also be treated as purely a rating by submitting a rating without comment text.
<input type="hidden" name="enableRating" value="true"/>

```
<input type="hidden" name="disableCookieCheck" value="true"/>
<input type="text" name="rating" value="4"/>
<input type="text" name="upvote" value="true" />
<input type="text" name="downvote" value="true" />
```

enableRating - needed to officially turn on ratings with a comment form

disableCookieCheck - overriding flag to turn off cookie check for duplicating ratings

rating - rating value from 1-3, 1-5, 1-10 depending on how the form is setup

Note: If no value is passed (such as if no option is selected in a radio button group), and comment text was specified, the comment will be created as a basic comment. If an invalid rating value is passed, or no rating was passed and no comment text was passed, the comment submission will error out, and the comment will not be created.

upvote/downvote - rating boolean when rating type is set to +1/-1

Note: If neither value is passed, or if both resolve to false, and comment text was passed the comment will be created as a BASIC comment. If neither value is passed, or if both resolve to false, and comment text was NOT passed, the submission will error out.

This is in addition to the comment form.

Additional Note with Respect to Threaded Comments

Ratings can never be made in threaded comment responses. The top-level item of a comment thread can be a rating, but all responses must be basic comments. If a user does submit a rating along with the 'parentID' parameter as a response, and the rating includes comment text, the item will be created as a basic comment. If there is no comment text, the submission will error out, and the comment will not be created.

To get the basic comments for a content object excluding ratings:	
ArrayList of Comment Objects	*$feedback.comments*
ArrayList of Comment Objects	$feedback.getComments(int contentItemID)
To get the ratings for a content object excluding comments:	
ArrayList of Rating Objects	$feedback.ratings()
ArrayList of Rating Objects	$feedback.ratings(int contentItemID)
To get all feedback for a content object:	
ArrayList of Feedback Objects	$feedback.allFeedback()
ArrayList of Feedback Objects	$feedback.allFeedback(int contentItemID)
To get a list of feedback sorted by average rating for a content object(200 max):	
ArrayList of SiteSearchResult Objects	$feedback.sortSearchResultsByAverageRating(ArrayList SiteSearchResult)
To get a list of content items sorted by avg rating for a content object(200 max):	
ArrayList of InstBase Objects	$feedback.sortContentByAverageRating(ArrayList InstBase)
To get average ratings and data about ratings (200 max):	

ArrayList of Rating Objects	$feedback.averageRating()
ArrayList of Rating Objects	$feedback.averageRating(int contentID)
Boolean	$feedback.isRated()
Boolean	$feedback.isRated(int contentID)
Rating Aggregate DataBean	$feedback.ratingData(int contentID)
Int	$feedback.ratingCount()
Int	$feedback.ratingCount(int contentID)
To get highest rated content objects:	
ArrayList of Highest Rated Objects	**$feedback.highestRated**(int interval, string intervalType, int destID, int numberOfResults)
ArrayList of Highest Rated Objects	**$feedback.highestRated**(int interval, string intervalType, int destID, int numberOfResults, date endDate)
ArrayList of Highest Rated Objects	**$feedback.highestRated**(int interval, string intervalType, int destID, int numberOfResults, int minimumNumberOfRatings)
ArrayList of Highest Rated Objects	**$feedback.highestRated**(int interval, string intervalType, int destID, int numberOfResults, date endDate, int minimumNumberOfRatings)
Given a Highest Rated object $hrItem:	
String	$hrItem.title
String	$hrItem.url
Integer	$hrItem.count
Integer	$hrItem.total
Integer	$hrItem.average
Integer	$hrItem.contentID
Integer	$hrItem.contentDestID

Given a Feedback object $feedbackItem:	$feedbackItem.attribute-name

Usage:

- **contentItemID** – the ID of the content item with the desired feedback (only required if not current item)
- **interval** – an integer value representing the number of days/months/weeks to go back from today's date in considering rating data
- **intervalType** – the units to treat the interval as, either D (days), M (months) or W (weeks)
- **destID** – destination ID
- numberOfResults – number of results to return
- **endDate** – date from which to calculate the interval from
- **minimumNumberOfRatings** – minimum number of ratings a content item needs to have in order to

be returned in results

Attributes:

- **feedbackLength** – the allowed length for the feedback's comment text field, honored when the feedback is submitted
- **feedbackPeriod** – the length of period in days for when this tag is used in the context of a content item (defined in the Content Type configuration)
- **feedbackPeriodType** – the type of feedback period, either D (day), W (week) or M (month)
- **ratingScoreType** – the type of rating score configured for the Content Type, either UP/DOWN, 1 to 3, 1 to 5, or 1 to 10

Code Examples:

For basic comments only:

```
#set($comments = $feedback.comments)
<ul>
#foreach($comment in $comments)
       <li>$comment.comment</li>
#end
</ul>
```

For ratings only:

```
#set($ratings = $feedback.ratings)
<ul>
#foreach($rating in $ratings)
       <li>$rating.rating | $rating.comment | $rating.ratingScoreType</li>
#end
</ul>
```

For comments and ratings combined:

```
#set($allFeedback = $feedback.allFeedback)
<ul>
#foreach($feedback in $allFeedback)
       <li>$feedback.comment | $feedback.rating (for rating type) |
$feedback.ratingScoreType</li>
#end
</ul>
```

For rating information:

```
#set($ratingScoreType = $feedback.ratingScoreType)
$ratingScoreType.description (example: "Ratings with values between 1-3")
$ratingScoreType.shortDescription (example: "1 to 3")
$ratingScoreType.maxRating (example: 3)
```

For sorting search results:

```
#set($mySearch = $cms.newSearch)
$mySearch.addKeywords($req.param("keywords"))
#set( $results = $mySearch.execute())
#set ($sortedResults = $feedback.sortSearchResultsByAverageRating($results))
```

For sorting $cms.links by rating:

```
#set ($sortedArticles =
$feedback.sortContentByAverageRating($cms.links(1,100)))
```

For displaying rating information:

```
Comment Average Rating: #if ($feedback.isRated()) $feedback.averageRating
#else Not Rated #end
```

For displaying a list of articles and their average ratings:

```
<ul>
#foreach($article in $sortedArticles)
```

```
       <li><a href="$article.url">$article.title</a>
       #if ($feedback.isRated($article.id))
              <strong>Rating: $feedback.averageRating($article.id)</strong>
       #else <strong>Not Rated</strong>
       #end<br/></li>
#end
</ul>
```

```
<ul>
#foreach($article in $sortedArticles)
       <li><a href="$article.url">$article.title</a></li>
       #if ($feedback.isRated($article.id))
              <strong>$feedback.ratingData($article.id).total /
$feedback.ratingData($article.id).count</strong>
       #end
</ul>
```

For displaying a list of articles and how many times it's been rated:

```
<ul>
#foreach($article in $sortedArticles)
       <li><a href="$article.url">$article.title</a>
       #if ($feedback.isRated($article.id))
              <strong>Count: $feedback.ratingCount($article.id)</strong>
       #end</li>
#end
</ul>
```

For displaying the highest rated items in a Most Popular List:

```
#set($mpItems = $feedback.highestRated(4,  "D", $cms.destination.ID, 5, 30))
<ul>
#foreach($mpItem in $mpItems)
       <li><a href="$mpItem.url">$mpItem.title</a> (Scaled:
$feedback.scaledRating($mpItem.average, 10)) , (Avg: $mpItem.average,
$mpItem.total/$mpItem.count)</li>
#end
</ul>
```

Threaded Comments

To get the threaded comments excluding ratings:	
ArrayList of Comment Objects	*$feedback.threadedComments*
ArrayList of Comment Objects	*$feedback.getThreadedComments (string topLevelSort)*
ArrayList of Comment Objects	*$feedback.getThreadedComments (string topLevelSort, string threadSort)*
ArrayList of Comment Objects	*$feedback.getThreadedComments (int contentId)*
ArrayList of Comment Objects	*$feedback.getThreadedComments (int contentId, string topLevelSort)*
ArrayList of Comment Objects	*$feedback.getThreadedComments (int contentId, string topLevelSort, string ThreadSort)*
To get the threaded comment ratings, and replies to those ratings (note top-level basic comments and their replies are excluded):	
ArrayList of Comment Objects	*$feedback.threadedRatings*
ArrayList of Comment Objects	*$feedback.getThreadedRatings (string topLevelSort)*
ArrayList of Comment Objects	*$feedback.getThreadedRatings (string topLevelSort, string threadSort)*
ArrayList of Comment Objects	*$feedback.getThreadedRatings (int contentId)*
ArrayList of Comment Objects	*$feedback.getThreadedRatings (int contentId, string topLevelSort)*
ArrayList of Comment Objects	*$feedback.getThreadedRatings (int contentId, string topLevelSort, string ThreadSort)*
To get all threaded comments including basic comments and ratings:	
ArrayList of Comment Objects	*$feedback.allThreadedFeedback*
ArrayList of Comment Objects	*$feedback.getAllThreadedFeedback (string topLevelSort)*
ArrayList of Comment Objects	*$feedback.getAllThreadedFeedback (string topLevelSort, string threadSort)*
ArrayList of Comment Objects	*$feedback.getAllThreadedFeedback (int contentId)*
ArrayList of Comment Objects	*$feedback.getAllThreadedFeedback (int contentId, string topLevelSort)*
ArrayList of Comment Objects	*$feedback.getAllThreadedFeedback (int contentId, string topLevelSort, string ThreadSort)*
To get the comment thread ID, parent comment id, and comment level (returns 0 for top-level and non-threaded comments):	
Int	*$feedback.threadID*
Int	*$feedback.parentID*
Int	*$feedback.level*

Usage:

- **contentItemID** – the ID of the content item with the desired comments (only required if not current item)
- **topLevelSort** – the sort specification for the order in which top-level comments should be sorted (if not specified they will be sorted in the order the comments were made)
- **threadSort** – the sort specification for the order in which comments should be sorted at each level within the thread hierarchy (if not specified they will be sorted in the order the comments were made)

A template accessing threaded comments should be able to access all of the fields from regular comments with some additions.These fields are only available in comments returned via the new Thread specific template tags.

In this table $threadedComment refers to a threaded comment returned from this sample code:

```
#foreach($threadedComment in $comments.threadedComments)
$threadedComment.level
#end
```

Given a Threaded Comment object $threadedComment:

Boolean	$threadedComment.hasResponses()
Boolean	$threadedComment.hasResponsesForAnyStatus()
Int	$threadedComment.numResponses
Int	$threadedComment.numResponsesForAnyStatus
Int	$threadedComment.numTotalResponses
Int	$threadedComment.numTotalResponsesForAnyStatus
Int	$threadedComment.rootCommentID
Date	$threadedComment.threadCreateDate
Date	$threadedComment.threadModifiedDate
Boolean	$threadedComment.isRejected()

Usage:

- **hasResponses** – returns true if the comment has 1 or more replies of status APPROVED or AUTHOR
- **numResponses** – number of comments of status APPROVED or AUTHOR made in direct reply to this thread; only includes direct children, not replies two or more levels below this comment in the thread hierarchy
- **numTotalResponses** – total number of descendant replies of status APPROVED or AUTHOR below this comment in the thread hierarchy
- **rootCommentID** – the comment ID of the top-level comment in this thread
- **threadCreateDate** – date of the top-level comment creation
- **threadModifiedDate** – date of the last reply in this thread
- **isRejected** – returns true if .status field returns something other than APPROVED or AUTHOR, otherwise false. Allows skipping or obfuscation of flagged, objectionable or otherwise rejected comments.

Post-back Javascript Function (for AJAX method only)

When using the AJAX submission utilities via #socialMediaCode and #socialMediaCodeNoLibrary macros, the new comment data will be populated to a wrapper javascript object: Clickability.Comment

The following fields may be set in the new object. Only the relevant fields will be set, if your comment form does not include a "name" field, for instance, name will remain null.

Variable	Description
Clickability.Comment.name	The given name of the comment submitter
Clickability.Comment.title	The title given to the comment
Clickability.Comment.email	The email address of the comment submitter
Clickability.Comment.website	The website address submitted by the commenter
Clickability.Comment.comment	The comment text
Clickability.Comment.flagged	A boolean specifying whether the comment has been flagged due to offensive content, or bot detection
Clickability.Comment.isAuthor	A boolean specifying whether the comment is an author comment
Clickability.Comment.status	possible status values: APPROVED, NOTAPPROVED, FLAGGED, AUTHOR, BOT, REJECTED
Clickability.Comment.error	The 'error' variable should be populated if an error occurred (i.e. if the form makes it to the returnFunction but nothing posts, an alert(error) can be done to see what is going wrong)
Clickability.Comment.id	The commentID of the newly added comment. Can be used as a parentID in future threaded responses to this comment
Clickability.Comment.threadID	The ID # of the comment thread, if the newly added comment was a threaded comment, made in response to another comment. 0 otherwise
Clickability.Comment.parentID	The comment ID of the parent comment of this comment, if the newly added comment was a threaded comment, made in response to another comment. 0 otherwise
Clickability.Comment.level	The zero-based thread level of the new comment. Top-level and non-threaded comments will have thread level 0, response to top level comments should have 1, etc.

Note: Only the relevant variables will be set; if your comment form does not include a “name” field, for instance, name will remain null. The ‘error’ variable should be populated if an error occurred (i.e. if the form makes it to the returnFunction but nothing posts, an alert(error) can be done to see what is going wrong).

If using AJAX, these JS variables can be used to determine if the comment was APPROVED or not and dynamically display the comment on the page. This is done via the function defined as the ‘handlerFunction’ argument in the postForm() call.

parentID Form Field

To create a threaded comment, add the form field 'parentID' to your form. If this field is in the form and the value designates the commentID of a valid and existing comment, a new threaded comment will be created as a response to the comment specified. If the 'parentID' field is not in the form, or the value does not designate a valid comment, the new comment will be created as a basic top-level comment.

In order for a threaded comment to be created as a response to another comment, the value of the parentID parameter must be an existing and valid commentID. The commentID may be obtained from the following sources

- From the $comment.ID or $comment.commentID field available when iterating over a comment list.
- From the Clickability.Comment.id field set after an AJAX comment submission using the #commentCode utilities.
- From the Comment.id field set after using the AJAX comment submission in the #socialMediaCode utilities.

Ratings with Threaded Comments

When using ratings in conjunction with threaded comments, only top-level comments can have ratings. Any responses will always be treated as Basic comments.

If a comment is submitted as a response and rating fields are passed as well, the following rules will apply:

- If comment text is submitted, the response will be created as a Basic comment. The rating score data will be ignored.
- If comment text is NOT submitted, the rating submission will error out and a response comment will not be inserted.

Sorting Threaded Comments

Two different sort specifications need to be specified when sorting the threaded comments for a content item:

1. The sort for the Top-Level comments.
2. The sort for the comments within a thread, within each level of the tree.

Any comment or threaded comment field may be applied to either of these sorts. However, these are the most common:

Top-Level Node Sort	Threaded Node Sort	Description
+commentDate	+commentDate	This will be the default, if a sort is not specified.
-commentDate	-commentDate	Most recently created top-level comments, but within each thread, the comments are displayed in comment order
+threadModifiedDate	+commentDate	Oldest comment threads first; within each thread comments are displayed in comment order
-threadModifiedDate	-commentDate	Most recently modified/replied to threads first; within each thread the comments are displayed in comment order

Example code to output threaded comments using <ul> and <li> tags

```
#set($lastLevel = 0)
<ul id="commentList" class="thread-level">
#foreach($comment in $comments.threadedComments)
	#if($comment.level < $lastLevel)
		#set($lastLevelMinusOne = $lastLevel - 1)
		#foreach($closeLevel in [$comment.level .. $lastLevelMinusOne]
)
			</ul></li>
		#end
	#end
	<li class="threaded-comment-row" id="comment.${comment.ID}">
	<div>(Displayable comment details go in this div) $comment.level -
$comment.comment</div>
	#if($comment.hasResponses())
		<ul class="thread-level">
	#else
		</li>
	#end
	#set($lastLevel = $comment.level)
#end
#if($lastLevel > 0)
	#set($lastLevelMinusOne = $lastLevel - 1)
	#foreach($lev in [0 .. $lastLevelMinusOne] )
		</ul></li>
	#end
#end
</ul>
```

Comments Search

FeedbackSearch Object	$cms.newFeedbackSearch - searches both comments and ratings
CommentSearch Object	$cms.newCommentSearch – searches only comments
RatingSearch Object	$cms.newRatingSearch - searches only ratings

By default, these search for all comments with APPROVED status of the selected feedback type, on the current domain.

Given a FeedbackSearch, CommentSearch or RatingSearch object $mySearch:

$mySearch.addKeywords(string search-keyword(s))

$mySearch.setContentID(integer content-id)

$mySearch.setContentTypeID(integer content-type-id)

$mySearch.setDomainID(integer domain-id)

$mySearch.setDestID(integer destination-id)

	$mySearch.setUserID(integer user-id)
	$mySearch.setRegUserID(integer user-id)
	$mySearch.setStatus(string status)
	$mySearch.setBeforeDate(string datefield, date date1)
	$mySearch.setAfterDate(string datefield, date date1)
	$mySearch.setDateRange(string date-field-name, date from, date to)
	$mySearch.setSearchField(string field-name)
	$mySearch.setSearchFields(arrayList field-names)
	$mySearch.setSearchFields(string comma-separated-field-names)
	$mySearch.sortByDate()
	$mySearch.page(integer n)
	$mySearch.pageSize(integer m)
	$mySearch.numPages

Usage:

- **search-keyword(s)** – the string to be searched for
- **datefield** – edit/editdate/modified/modifieddate, create/createdate/commentdate
- **field-names** – all, comment, name, email, title, contenttitle, website, ip, rankusername, rankemail, rankipaddress
- **status** – the status of the comment (BLOCKED, NOTAPPROVED, APPROVED, FLAGGED, AUTHOR, BOT, BANNED, REJECTED, MULTIPLE, DOWNRANKED, DELETED, INITIAL)

ArrayList of CommentSearchResults

	$myCommentSearch.execute

Given a CommentSearchResult object $searchResult:

CommentType	$searchResult.commentType – returns “Basic” for comments or “Rating” for ratings
Boolean	$searchResult.flagged
String	$searchResult.title
String	$searchResult.comment
String	$searchResult.name
String	$searchResult.website
String	$searchResult.IPAddress
String	$searchResult.contentTitle
Int	$searchResult.contentID
Date	$searchResult.commentDate
Date	$searchResult.modifiedDate

String	$searchResult.status
Int	$searchResult.numOfNaughtyWords
Int	$searchResult.commentID
Int	$searchResult.ID
Rating Specific:	
Int	$searchResult.rating
String	$searchResult.ratingScoreType
ThreadedComment Specific:	
Int	$searchResult.threadID
Int	$searchResult.parentID
Int	$searchResult.level

Code Example:

```
#set($feedbackSearch = $cms.newFeedbackSearch)
$feedbackSearch.addKeywords("great")
$feedbackSearch.setDestID(12345)
$feedbackSearch.setSearchField("name")
$feedbackSearch.pageSize(50)
$feedbackSearch.setContentTypeID(23456)
#set($feedbackResults = $feedbackSearch.execute)
There are $feedbackResults.size() results.<br>
#foreach($comment in $util.sort($feedbackResults, "-commentDate")) ##sorting
by newest to oldest
    comment date: $util.date("MMM d, yyyy h:mm a", $comment.commentDate)<br>
    comment: $comment<br>
    comment name: $comment.name<br>
    comment email: $comment.email<br>
    content id: $comment.contentID<br>
    content title: $comment.contentTitle<br>
#end
```

Formatting Text

Formatting

String	$util.text(string htmltext)
String	$util.html(string plaintext)
String	$util.textWidth(string text, int width)

Usage:

- **htmltext** – The HTML text to be reformatted as plain text.
- **plaintext** – The plain text to be reformatted as HTML.
- **width** – The number of characters per line to restrict the text element to.

These tags are most often used in conjunction with content tags. The first two formatting functions allow for the conversion of HTML to plain text, and vice versa. The following actions are performed:

```
$util.text
```

- strips all HTML, Javascript, and other <tags>
- turns paragraphs tags (<P>) into 2 New Lines
- turns line breaks (<br/>) into 1 New Line
- turns   into a space character

```
$util.html
```

- turns New Lines into <br /> tags

The $util.textWidth tag allows for a text content element's width to be restricted, such as for use in creating a plain text newsletter, in which you would like to automatically put in line breaks to prevent wrapping inconsistencies in various mail clients.

Examples:

```
$util.text($cms.content.summary)
```

- This would strip all extra formatting out of the summary element of the content.

```
$util.html($cms.content.body)
```

- This would insert <br /> tags to the body in place of any New Line characters.

```
$util.textWidth($util.text($cms.content.summary),70)
```

- This would restrict the line width to 70 characters for the display of the plain text version of the summary element of the content.

  Note: $util.textWidth always inserts a <br /> at the end of the line – even if the element width is already less than the designated limit.

Paragraphs

ArrayList of Strings	$clickabilityItem.paragraphs(string *content-field-tag*)
String	**$util.paragraph**(string *text*, int *start-index*, int *number-paragraphs*)
String	**$util.textParagraph**(string *text*, int *start-index*, int *number-paragraphs*)

Usage:

- **content-field-tag** – the tagname for the clickabilityItem property whose returned value you want to format as paragraphs
- **text**– The text to be split into paragraphs.
- **start-index** – The index number of the first paragraph to return.
- **number-paragraphs** – The total number of paragraphs to return.
- **$clickabilityItem** – a content item (obtained via $cms.content or other means), mailing item ($cms.mailing), newsletter item ($cms.newsletter), or websiteSection ($cms.websiteSection)

The paragraphination functions split a text body into paragraphs. The specific function to use depends on what you're trying to do, and what type of text you are using (HTML or plain text).

Obtain All Paragraphs from a Given Field

If you want to retrieve ALL paragraphs from a field of a Clickability Platform object (such as a Content, WebsiteSection, Newsletter or Mailing field), use $clickabilityItem.paragraphs to get an ArrayList of the each paragraph's contents (*without* <p> tags) .

Example

Given a content item that has a Body property with tagname 'body':

```
$cms.content.paragraphs("body")
```

- would return an ArrayList of every paragraph in the content's body field (without the <p> tags), which might then be iterated for display as follows:

```
#foreach($p in $cms.content.paragraphs("body"))
       <p>$p</p>
#end
```

You can also use the ArrayList returned by this tag to determine how many paragraphs it contains:

```
$cms.content.paragraphs("body").size()
```

Obtain a Single Paragraph or Subset of Parargraphs

If the intention is to retrieve only a specific paragraph or group of paragraphs, you can use the below methods to zero in on just those paragraphs of interest. Note you must know what type of text you are passing in (html text w/ <p> tags, or plain text that uses line breaks to indicate paragraphs).

- `$util.paragraph` – splits into paragraphs wherever it finds paragraph (<p>) tags, so use this with HTML-formatted text. The string that is returned includes the <p> tag formatting.

- `$util.textParagraph` – splits into paragraphs wherever it finds two consecutive New Lines, so use this with plain text. The string that is returns includes the NewLine characters (but no <p> tags – so if you need to display as HTML run it through the $util.html() tag explained earlier in this chapter).

Examples:

```
$util.paragraph($cms.content.body,1,1)
```

- This would return the first paragraph of the body (assuming it's an HTML field).

```
$util.paragraph($cms.content.body,2,5)
```

- This would return the second through the sixth paragraphs of the body (assuming it's an HTML field).

```
$util.textParagraph($cms.content.summary,1,1)
```

- This would return the first paragraph of the summary (assuming it's a plain text field).

Pagination

String	**$util.page**(string text, int page-number, int paragraphs-per-page)
String	**$util.textPage**(string *text*, int *page-number*, int *paragraphs-per-page*)
Integer	**$util.countPages**("text element", paragraphs-per-page)
Integer	**$util.countTextPages**("text element", paragraphs-per-page)

Usage:

- **text** – The text to be split into paragraphs.
- **page-number** – Which page to return
- **paragraphs-per-page** – The number of paragraphs per page.

The pagination functions split a text body into pages. The specific function to use depends on the type of text used (either HTML or plain text).

- `$util.page` – splits into pages based on paragraphs designated by tags (<P>)
- `$util.textPage` – splits into pages based on paragraphs designated by two consecutive New Lines.

These tags are most often used in conjunction with content tags.

The following two tags will return the number of pages based on the specified paragraphs per page:

```
$util.countPages("text element", paragraphs-per-page)

$util.countTextPages("text element", paragraphs-per-page)
```

Examples:

```
$util.page($cms.content.body,1,5)
```

- This would return the first page of the body (assuming it's an HTML field) – allotting five paragraphs per page.

```
$util.page($cms.content.body,2,15)
```

- This would return the second page of the body (assuming it's an HTML field) - allotting fifteen paragraphs per page.

```
$util.textParagraph($cms.content.summary,2,5)
```

- This would return the second paragraph of the summary (assuming it's a plain text field) - allotting five paragraphs per page.

Translating text

String	$cms.translate(string *text*)

Usage:

- **text** – Any string returned from a content element or other source.

Typically, when you place a text element onto a page it **IS NOT** parsed and executed by the publishing engine. However, cases arise when it is desirable to have the publisher "translate" the text.

The $cms.translate tag, tells the publisher to parse text and execute all template language tags found in the text. This can be used when you include template tags and code within individual content items, include files, external files, or other sources.

Examples:

Translating tags contained in Include files:
Normally, the contents pulled in by the $cms.include() tag are included just as text that is not parsed by the publishing engine. But by passing the Include text throught $cms.translate, like this:

```
$cms.translate($cms.include("header.js").text)
```

any Velocity tags contained in the include will be parsed.

Translating tags contained in content items:
It is generally not advisable to incorporate template code into content items, since the content types should have fields for each of the pieces needed to flexibly configure the template. However, we recognize special cases.

For example, if you have one content item to be published across 10 sites, with the same exact content except for a varying phrase that must appear in the middle of the body text.

If the body field of a content item contained the following text:

```
The $cms.websiteSection("/").name contest can be found here
```

then this line of template code:

```
$cms.translate($cms.content.body)
```

would output:

- The {top-level website section's name} contest can be found here

Encoding Options

String	$util.encode.html(string *text*)
String	$util.encode.unHTML(string *text*)
String	$util.encode.htmlEntityToText(string *text*)
String	$util.encode.form(string *text*)
String	$util.encode.textarea(string *text*)
String	$util.encode.js(string *text*)
String	$util.encode.url(string *text*)
String	$util.encode.unURL(string *text*)
String	$util.encode.uriEncode(string *scheme,* string *text*)
String	$util.encode.xml(string *text*)
String	$util.SHA1AsBase64(string *text*)
String	$util.SHA1AsHex(string *text*)
String	$util.MD5AsHex(string *text*)
String	$util.MD5AsBase64(string *text*)
String	$util.hmacSha1AsBase64(string *text,* keystring *text*)
String	$util.hmacSha256AsBase64(string *text,* keystring *text*)
String	$util.hmacMd5AsBase64(string *text,* keystring *text*)
String	$util.hmacSha1AsHex(string *text,* keystring *text*)
String	$util.hmacSha256AsHex(string *text,* keystring *text*)
String	$util.hmacMd5AsHex(string *text,* keystring *text*)
String	$util.encode.xssHtml(string *htmlText*)
String	$util.encode.xssAttrib(string *attribText*)
String	$util.encode.xssJs(string *jsText*)
String	$util.encode.url(string *paramNameOrValue*)
String	$util.encode.xssStrictHtml(string *htmlText*)
String	$util.encode.xssRemoveHtml (string *htmlText*)
String	$util.base64Encode(string *text*)
String	$util.base64Decode(string *text*)
String	$util.encode.smartQuotesToPlainText(string *text*)

Usage:

- text – The text to be encoded.
- scheme – http or https

At times in web development it is beneficial to encode or decode string. The following functions and conversions are available:

HTML encoding

```
$util.encode.html(string text)
```

From	To
&	&amp;
<	&lt;
>	&gt;
"	&quot;
\n	
\r\n	

```
$util.encode.unHTML(string text)
```

From	To
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'
 	\n

```
$util.encode.htmlEntityToText(string text)
```

From	To
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'

Form Encoding

```
$util.encode.form(string text)
```

From	To
&	&amp;
"	&quot;

```
$util.encode.textarea(string text)
```

From	To
&	&amp;
"	&quot;
<	&lt;
>	&gt;

Javascript Encoding

```
$util.encode.js(string text)
```

From	To
\	\\
'	\'
"	\"

URL Encoding

```
$util.encode.url(string text)
```

- The alphanumeric characters "a" through "z", "A" through "Z" and "0" through "9" remain the same.
- The special characters ".", "-", "*", and "_" remain the same.
- The space character " " is converted into a plus sign "+".
- All other characters are unsafe and are first converted into one or more bytes using some encoding scheme. Then each byte is represented by the 3-character string "%xy", where xy is the two-digit hexadecimal representation of the byte. The recommended encoding scheme to use is UTF-8. However, for compatibility reasons, if an encoding is not specified, then the default encoding of the platform is used.

```
$util.encode.unURL(string text)
```

- The alphanumeric characters "a" through "z", "A" through "Z" and "0" through "9" remain the same.
- The special characters ".", "-", "*", and "_" remain the same.

- The plus sign "+" is converted into a space character " ".
- A sequence of the form "%xy" will be treated as representing a byte where xy is the two-digit hexadecimal representation of the 8 bits. Then, all substrings that contain one or more of these byte sequences consecutively will be replaced by the character(s) whose encoding would result in those consecutive bytes. The encoding scheme used to decode these characters may be specified, or if unspecified, the default encoding of the platform will be used.

```
$util.encode.uriEncode(string scheme, string text)
```

- The alphanumeric characters "a" through "z", "A" through "Z" and "0" through "9" remain the same.
- The special characters ".", "-", "*", and "_" remain the same.
- The space character " " is converted into "%20".

Example:

```
$util.encode.uriEncode("http", "//media.yourdomain.com/images/This Is A
Test.jpg")

would result in the output,

"http://media.yourdomain.com/images/This%20Is%20A%20Test.jpg"
```

XML Encoding

```
$util.encode.xml(string text)
```

From	To
&	&amp;
<	&lt;
>	&gt;
'	&apos;
"	&quot;

Hash Encoding

```
$util.SHA1AsBase64(string text)
$util.SHA1AsHex(string text)
```

- These will output the SHA-1 encoded string as either Hex or Base64.

```
$util.MD5AsBase64(string text)
$util.MD5AsHex(string text)
```

- These will output the MD5 hashed value of the given string.

```
$util.hmacSha1AsBase64(string text, keystring text)
$util.hmacSha256AsBase64(string text, keystring text)
$util.hmacMd5AsBase64(string text, keystring text)
$util.hmacSha1AsHex(string text, keystring text)
$util.hmacSha256AsHex(string text, keystring text)
$util.hmacMd5AsHex(string text, keystring text)
```

- The HMAC versions output the SHA-1, SHA-256 or MD5 hashed value as either Hex or Base64. **Note:** The tag `$util.HMACAsBase64` still returns a Base64 string but has been deprecated.

XSS Remediation

> ☆ **Cross-Site Scripting (XSS) Prevention is considered a best practice.**
>
> **For more details and more best practices, go to**
> *http://community.clickability.com* **> Developer Forum > Technical Docs & Webinars.**

The following template tags are provided for the types of encoding described in the OWASP document at http://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet. These template tags are based on the functionality provided by the OWASP ESAPI.

`$util.encode.xssHtml(string htmlText)`

- Encodes text for use between HTML tags using HTML entity encoding. This pertains to rule #1 in the OWASP document.
- This is not to be used inside of actual tags such as in an attribute value, and attribute name, or a tag name itself. This should also never be used inside of a <script> tag, as the xssJs utility should be used instead.
- **Example:**
  - $util.encode.xssHtml($req.param("someParam"))

`$util.encode.xssAttrib(string attribText)`

- Encodes text for use in an HTML attribute value. This pertains to Rule #2 in the OWASP document.
- This should also be use for encoding values in form tags, such as the `<input>` tag, as these values are specified in attribute values. This should not be used for encoding attribute names. This should not be used in attributes which are JavaScript event handlers, in those cases use the `xssJs` method.
- **Example:**
  - <input type="text" name="data" value="$util.encode.xssAttrib($req.param("val"))">

`$util.encode.xssJs(string jsText)`

- Encodes text for use inside of a data value or function argument in JavaScript. Replacements should never be made in JavaScript code itself, as no amount of encoding can prevent attacks there. This pertains to Rule #3 in the OWASP document.
- **Examples:**
  - Inside a string used as a function argument:
    o <script>alert('$util.encode.xssJs($req.param("val")')</script>
  - Inside a string used as a variable value:
    o <script>var myVar = '$util.encode.xssJs($req.param("val")';</script>
  - In a quoted value in an event handler attribute:
    o <a onClick="var myVar='$util.encode.xssJs($req.param("val1")';
    alert('$util.encode.xssJs($req.param("val2")')">JS Anchor

`$util.encode.url(string paramNameOrValue)`

- Encodes text for use in a URL name or parameter. This pertains to rule #5 in the OWASP document.
- **Example:**

- Submit

```
$util.encode.xssStrictHtml(string htmlText)
```

- Sanitizes HTML text for direct display in an HTML page. This uses a very strict policy for removing any potentially insecure tags.
- Only the following tags will be retrained in the output:
  o <b> <i> <p>
 <a>   <dl> <dt> <dd>   <tt> <blockquote> <ecode> <quote>

```
$util.encode.xssRemoveHtml(string htmlText)
```

- Strips all HTML tags out of the input, leaving only the raw text outside of or between tags.

Miscellaneous

```
$util.base64Encode(string text)
$util.base64Decode(string text)
```

- These will encode or decode a string as Base64.

```
$util.smartQuotesToPlainText(string text)
```

- Converts a string encoded as windows-1252 or including commonly occuring unicode to valid ascii/iso-8859-1 characters. This is useful for email subjects where transfer occurs as 7bit. The list of codes converted follows:

From (Encoded)	To (ASCII text)
\u0085	"..."
\u2026	"..."
\u0086	"*" (dagger character)
\u0091	"\"
\u2018	"\"
\u0092	"\"
\u2019	"\"
\u0093	"\"
\u201C	"\"
\u0094	"\"
\u201D	"\"
\u0095	"*" (bullet point)
\u0096	"-"
\u2013	"-"
\u0097	"-"

\u2014	"-"
\u0099	"TM"
\u00a0	" "

Using Placement Lists

Getting Placement Lists

ContentObject	**$cms.link**(int *index*)
ContentObject	**$cms.link**(string websiteSection-name|website-path, int index)
ContentObject	**$cms.link**(string domain, string websiteSection-name|website-path, int index)
ArrayList of ContentObjects	$cms.links
ArrayList of ContentObjects	**$cms.links**(int start-index, int list-size)
ArrayList of ContentObjects	**$cms.links**(string websiteSection-name|website-path, int start-index, int list-size)
ArrayList of ContentObjects	**$cms.links**(string domain, string websiteSection-name|website-path, int start-index, int list-size)

Usage:

- **website-section-name** – The name of the placement list from which to retrieve a set of content items.
- **website-path** – The URL path to the website section from where the placement-list will be taken from. When omitted, the publisher defaults to the name of the website section of the item or page currently being published.
- **index** – Which content item to get – 3 indicates to return the 3rd item in the list of all available links.
- **start-index** – Which content item to start with – 3 indicates start with the 3rd item in the list of all available links.
- **list-size** – The maximum number of links to return. The size of the returned list may be smaller than this number if not enough links are available. List-size can only be used if start-index is specified as well.

Defaults:

- **website-section-name** – the placement list associated with the website-section of the item or page currently being published.
- **start-index** – first item
- **list-size** – 10 items

The *links* tags allow access to content that has been placed or targeted on a site. Placement Lists must be accessed with the #foreach tag as the results are returned as an ArrayList of content item objects. An exception to this is if the link (singular) tag is specified, in which case a single content object is accessible directly.

IMPORTANT:
- The default list-size is 10 items. If no list-size is specified, only 10 items will be returned.
- The maximum number of items that can be retrieved via Placement Lists is 200 items.

Links Examples:

```
$cms.link(4).title
```

- Returns the title of the 4th item targeted to the website section beign published.

```
$cms.link("News",1).url
```

- Returns the url of the 1st item targeted to the “News” website section.

```
#foreach($article in $cms.links)
	<a href="$article.url">$article.title</a><br/>
#end
```

- Iterates items 1 through 10 (see Defaults) in the website section being published, displaying the title of each.

```
#foreach($article in $cms.links(5))
	<a href="$article.url">$article.title</a><br/>
#end
```

- Iterates items 5 through 15 (10= default list-size) in the website section being published, displaying title of each.

```
#foreach($article in $cms.links(5,20))
	<a href="$article.url">$article.title</a><br/>
#end
```

- Iterates items 5 through 25 in the website section being published, displaying the title of each.

```
#foreach($article in $cms.links("News"))
	<a href="$article.url">$article.title</a><br/>
#end
```

- Iterates items 1 through 10 (see Defaults) in the “News” website section, displaying the title of each.

```
#foreach($article in $cms.links("News",5,20))
	<a href="$article.url">$article.title</a><br/>
#end
```

- Iterates items 5 through 25 in the “News” website section, displaying the title of each.

WARNING: When retrieving a placement list using the “links” (not “link”) command, it is advisable to use a #foreach loop. This will prevent the template from breaking if there are no available links to return.

Filtering Placement Lists on Content Type

ArrayList of ContentObjects	**$cms.filteredLinks**(string *contentType*)
ArrayList of ContentObjects	**$cms.filteredLinks**(string *contentType,* int *start-index,* int *list-size*)
ArrayList of ContentObjects	**$cms.filteredLinks**(string contentType, string websiteSection-name|website-path, int start-index, int list-size)
ArrayList of ContentObjects	**$cms.filteredLinks**(string contentType, string domain, string websiteSection-name|website-path, int start-index, int list-size)

Usage:

- **contentType** – Limit the results to a specific content type (ie. sports articles)
- **domain** – The domain of the site where the placement list exists
- **website-section-name** – The name of the placement list from which to retrieve a set of content items.
- **website-path** – The URL path to the website section from where the placement-list will be taken from.

When omitted, the publisher defaults to the name of the website-section of the item or page currently being published.

- **start-index** – Which content item to start with – 3 indicates start with the 3rd item in the list of all available links.
- **list-size** – The maximum number of links to return. The size of the returned list may be smaller than this number if not enough links are available. List-size can only be used if start-index is specified as well.

Defaults:

- **domain** – the domain currently being published
- **website-section-name** – the placement list associated with the website-section of the item or page currently being published.
- **start-index** – first item
- **list-size** – 10 items

The filteredLinks tag allows the additional capability of specifying particular content types to be included in the list of links.

IMPORTANT:
- The default list-size is 10 items. If no list-size is specified, only 10 items will be returned.
- The maximum number of items that can be retrieved via Placement Lists is 200 items.

Filtered Links Examples:

```
#foreach($article in $cms.filteredLinks("Sports Article", "Sports",1,5))
      <a href="$article.url">$article.title</a><br/>
#end
```

Using Categories in Placement Lists

$cms.getCategorization can be used to define categorization criteria that can then be used to retrieve content from placement lists. For more details on defining the criteria, see Section II of the "Categories" chapter: "Defining Categorization Criteria".

CategorizationCriteria Object	**$cms.getCategorization**(ArrayList *category-sets*, ArrayList *categories*, boolean *match-all?*)
ArrayList of ContentObjects	**$cms.categorizedLinks**(CategorizationCriteriaObject *cat-obj*, int *start-index*, int *list-size*)
ArrayList of ContentObjects	**$cms.categorizedLinks**(CategorizationCriteriaObject *cat-obj*, string *website-section-name*|*website-path*, int *start-index*, int *list-size*)
ArrayList of ContentObjects	**$cms.categorizedLinks**(CategorizationCriteriaObject *cat-obj*, string *domain*, string *website-section-name*|*website-path*, int *start-index*, int *list-size*)

Usage:

- **cat-obj** – The object which specifies the category restrictions on the links to be returned.
- **category-sets** – a list of category sets either all by name (e.g.["SetA","SetB","SetC"...])or all by ID (e.g.[id1,id2,id3…])
- **categories** – a list of category values (one for each entry in the categorySets list, and in the same order), either by name (ex: ["CatNameFromSetA","CatNameFromSetB"...]) or by ID (ex:[CatIDFromSetA, CatIDFromSetB...])=
- **match-all?** – Specifies if the categorization criteria is to match all constraints (true) or any constraint (false).
- **domain** – The domain of the site where the placement list exists
- **website-section-name –** The name of the placement list to retrieve a set of content items.
- **website-path** – The URL path to the website section from where the placement-list will be taken.

When omitted, the publisher defaults to the name of the website section of the item or page currently being published.

- **start-index** – Which content item to start with – 3 indicates start with the 3rd item in the list of all available links.
- **list-size** – The maximum number of links to return. The size of the returned list may be smaller than this number if not enough links are available. List-size can only be used if start-index is specified as well.

Defaults:

- **domain** – the domain currently being published
- **website-section-name** – the placement list associated with the website section of the item or page currently being published.
- **start-index** – first item
- **list-size** – 10 items
- **match-all?** – true

Use the categorizedLinks tag to get the links conforming to your categorization criteria object.

IMPORTANT:
- The default list-size is 10 items. If no list-size is specified, only 10 items will be returned.
- The maximum number of items that can be retrieved via Placement Lists is 200 items.

Examples:

Consider the following two Categorization sets:

Category Set #1:
Region:
- East
  - North East
  - Mid-Atlantic
- West
  - Pacific Northwest
  - Calif
    - NorCal
    - SoCal
- North
  - Great Lakes
  - Rockies
- South
  - Texas
  - Gulf Coast

Category Set #2:
Source:
- AP
- UP

To get all of the links categorized in the West Region:

```
#set($cat1 = $cms.getCategorization(["Region"],["West"]))
#foreach($link in $cms.categorizedLinks($cat1))
```

To get all of the links categorized in the California Region:

```
#set($cat1 = $cms.getCategorization(["Region"],["West/Calif"]))
#foreach($link in $cms.categorizedLinks($cat1))
```

To get all of the links categorized in the NorCal Region:

```
#set($cat1=$cms.getCategorization(["Region"],["West/Calif/NorCal"]))
#foreach($link in $cms.categorizedLinks($cat1))
```

To get all of the links categorized in the North Region **AND** in the AP Source:

```
#set($cat1 = $cms.getCategorization(["Source","Region"],["AP","North"]))
#foreach($link in $cms.categorizedLinks($cat1))
```

To get all of the links categorized in the North Region **OR** in the AP Source:

```
#set($cat1 = $cms.getCategorization(["Source","Region"],["AP","North"],
true))
#foreach($link in $cms.categorizedLinks($cat1))
```

Advanced Placement Lists

LinksObject	$cms.newLinks
Given a Links object $myLinks:	
	$myLinks.addContentType(string *contentType*|int *contentTypeID*)
	$myLinks.excludeContentType(string *contentType*| int *contentTypeID*)
	$myLinks.setCategorization(categorizationCriteriaObject *cat-obj*)
	$myLinks.setDestination(string *website-section-name*|*website-path*)
	$myLinks.setDestination(string domain|int domainID, string website-section-name|website-path)
ArrayList of ContentObjects	$myLinks.links
ArrayList of ContentObjects	**$myLinks.links**(int *startIndex*)
ArrayList of ContentObjects	**$myLinks.links**(int *startIndex*,int *list-size*)

Usage:

- **cat-obj** – The object which specifies the category restrictions on the links to be returned.
- **domain or domainID** – Limit the results to a specific domain (by name or id)
- **contentType or contentTypeID** – Limit the results to a specific content type (by name or id)
- **website-section-name** – The name of the placement list from which to retrieve a set of content items.
- **website-path** – The URL path to the website section from where the placement-list will be taken from.

When omitted, the publisher defaults to the name of the website section of the item or page currently being published.

- **start-index** – Which content item to start with – 3 indicates start with the 3rd item in the list of all available links.
- **list-size** – The maximum number of links to return. The size of the returned list may be smaller than this number if not enough links are available. List-size can only be used if start-index is specified as well.

Defaults:

- **domain** – the name or id of the domain currently being published
- **website-section-name | website-path** – the placement list associated with the website-section of the item or page currently being published.
- **start-index** – first item
- **list-size** – 10 items

The advanced placement list tags extend the functionality provided in previous sections by allowing more control over the links criteria and by adding the ability to combine the functions together.

Create the Links Object

```
#set($myLinks=$cms.newLinks)
```

Specify Content Types

```
$myLinks.addContentType(string contentTypeName| int contentTypeID)
$myLinks.excludeContentType(string contentTypeName| int contentTypeID)
```

The default behavior is to accept all content types. You may filter these results by either including one or more content type, or by excluding one or more content type. By adding content types to the links list filter, all previously defined exclusions are removed. Likewise, by excluding content types, all previous inclusions are removed.

Specify Categorization

```
$myLinks.setCategorization(CategorizationCriteriaObject catObj)
```

The default behavior is to accept all categories of content items. A categorization criteria object can be provided to limit these results. See the previous section for details on how to construct categorization criteria objects.

Specify Destination

```
$myLinks.setDestination(string website-section-name|website-path)
$myLinks.setDestination(string domainName, string website-section-
name|website-path)
$myLinks.setDestination(string domainID, string website-section-
name|website-path)
```

The default behavior is to look for content items in the destination (website section) being published. If you would like to draw content targeted to an alternative destination, it can be referenced by destination name or id (you must include the domain name if that destination is on a different domain).

Note: content will only be drawn from ONE destination, so setDestination() should be called only once per links list.

Retrieve the Links

```
$myLinks.links
$myLinks.links(int startIndex)
$myLinks.links(int startIndex,int size)
```

The results of the links method are returned as an ArrayList of ContentObjects. Both the starting point of the list and the size of the list can be specified as desired, but if not specified will default to the values noted in the table above.

Example

The following example would display the titles of the 3rd to 13th News or Sports articles, targeted to the “Home Page”, and categorized as “West/Calif/NorCal” in the “Region” category set:

```
#set($myLinks=$cms.newLinks)
#set($cat1=$cms.getCategorization(["Region"],["West/Calif/NorCal"]))
```

```
$myLinks.addContentType("News Story")
$myLinks.addContentType("Sports Story")
$myLinks.setDestination("Home Page")
$myLinks.setCategorization($cat1)
#foreach($item in $myLinks.links(3,10))
      $item.title  <br/>
#end
```

Categories

Getting the Categorization of a Content or Media Item

As we saw in the "Content" chapter, you can derive the categorization settings of a specific content item by calling getCategorization() on that item. The same is true of MediaObjects.

To get a Categorization object $catgzn

Categorization Object	$contentItem.categorization
Categorization Object	$mediaObject.categorization

Given a Categorization object $catgzn:

ArrayList of CategorySet Objects	$catgzn.sets
ArrayList of Ints	$catgzn.setIDs
Boolean	**$catgzn.allowsMultiple**(int *setID* or String *setName*)
ArrayList of Category Objects	**$catgzn.getCategories**(int *setID* or String *setName*)
Category Object	**$catgzn.getCategory**(int *setID* or String *setName*)

Usage:

- **$contentItem** – a content item object, obtained through a $cms.content tag or other means.
- **$mediaObject** – a MediaObject, obtained through a $cms.media tag or other means
- **setID** – the id of the CategorySet of interest
- **setName** – the name of the CategorySet of interest

The CategorizationObject that is returned can then be used to derive the relevant CategorySet and Category objects, from which additional categorization details (such as name, level, parent) can be obtained using the methods described in Section V of this chapter.

Example

```
#set($content = $cms.content)
$content.title<br/>

#set($catgzn = $content.categorization)  ##creates an object that represents
all categories for the content item

#set($catSetIDArray = $util.newArrayList)
#set($catSetIDArray = $catgzn.setIDs) ##returns an array of all category set
IDs

#foreach($catSetID in $catSetIDArray) ## loops through all sets and prints
out the cat set name and the category name

      #if($content.categorization.getCategory($catSetID).name != "" &&
$content.categorization.getCategory($catSetID).name != "Uncategorized") ##
check for null or uncategorized
```

```
            $cms.getCategorySet($catSetID).name :
$content.categorization.getCategory($catSetID).name <br/><br/>
      #end
#end
```

Defining Categorization Criteria

$cms.getCategorization is used to define categorization criteria that can then be used in retrieving content from placement lists (see the "Placement Lists" chapter) or performing search queries (see the "Site Search" chapter).

Categorization Criteria Object	**$cms.getCategorization**(ArrayList of*categorySets,* ArrayList of*categoryValues [*, Boolean *match-all?]*) Given a CategorizationCriteria object **$catCriteria** • $catCriteria.setExcludeChildren(ArrayList setNamesOrIDs) Given multiple single category CategorizationCriteria objects **$singleCat1 and $singleCat2** • $singleCat2.setMatchAny()

Usage:

- **categorySets** – a list of category sets either all by name (e.g.["SetA","SetB","SetC"...])or all by ID (e.g.[id1,id2,id3…])
- **categoryValues** – a list of category values (one for each entry in the categorySets list, and in the same order), either by name (ex: ["CatNameFromSetA","CatNameFromSetB"...])or by ID (ex:[CatIDFromSetA, CatIDFromSetB...])
- **match-all?** – Specifies if the categorization criteria is to match all constraints (true) or any constraint (false)
- **setNamesOrIDs** – An arraylist of the Sets for which you want to exclude child categories from the matching criteria.

Default:

- **match-all?** – true

Constructing the Criteria Arrays

The categorySets and categoryValues lists must be specified such that:

The 1st category in the categoryValues list represents an acceptable value for the 1st category set in the categorySets list, the 2nd category for the 2nd category set, and so on. (This means the 2 arraylists will contain an equal number of items).

All arraylist entries are specified by name (string), or all are specified by ID (integer); you cannot, for instance, give the categorySets arraylist as names and the categoryValues arraylist as IDs.

Sub-category values are specified using the full "path". For example:

```
["CatFromSet1/CatFromSet1SubCat"]
or ["/CatFromSet1/CatFromSet1SubCat"]
```

Examples:

```
$cms.getCategorization(["Region","Station"], ["West/California", "KXYZ"])
```

- will return a CategorizationCriteriaObject that will match content categorized as Region=West/California, and Station=KXYZ.

```
$cms.getCategorization(["Article Type", "Article Type", "Station"],
["Sports", "News", "KXYZ"])
```

- will return a CategorizationCriteriaObject that will match content categorized as Article Type=Sports and Article Type = News and Station=KXYZ.

Match-All?

The Match-All? boolean option sets the matching method for the categorization criteria. The default behavior (Match-All? = true) is to enforce matching of ALL the categorization criteria. The examples above will only return content where ALL conditions are true.

To switch the match method to require matching on *any* of the categorization criteria, set Match-All? to false. **For example,** the following:

```
$cms.getCategorization(["Article Type", "Article Type", "Station"],
["Sports", "News", "KXYZ"], false)
```

- would match content categorized as *either* Article Type=Sports *or* Article Type = News *or* Station=KXYZ.

Enforcing Strict Category Matching

By default, the categorization criteria you define will (when passed into a search or a placement list) return not only content categorized as the specified category value, but ALSO content categorized as any of the subcategories of the specified value.

To prevent this from happening, (i.e. to return ONLY content categorized as the exact category specified), call setExcludeChildren()on the CategorizationCriteriaObject before passing it to a search or placement list.

For example:

```
#set($catgzn = $cms.getCategorization(["Area"],["West"]))
$catgzn.setExcludeChildren(["Region"])

#foreach($item in $cms.categorizedLinks($catgzn))
$item.title<br/>
#end
```

- would return only those content items which are categorized as Area = West. It would NOT return content categorized as Area = West/California.

Creating a Category Histogram

ArrayList of HistogramResult objects	$cms.newCategoryHistogram(string *catSetName*)
Given a Histogram object $histogram:	
	$histogram.addWSS()
	$histogram.addWSS(int *wssID* [, Boolean *includeDescendants*])
	$histogram.addWSS(string *wssPath* [,Boolean *includeDescendants*])
	$histogram.addWSS(WebsiteSection *wssObj* [, Boolean *includeDescendants*])
	$histogram.setLevel(int *level*)
	$histogram.setParentID(int *catID*)
ArrayList of HistogramResult objects	$histogram.execute()
Given a HistogramResult object $result:	
	$result.categoryName or $result.name
	$result.categoryID or $result.ID
	$result.setID
	$result.count

Usage:

- **catSetName** – The name of a category set defined in the Clickability Platform
- **wssID** – The id of the website section from which matching content items should be returned.
- **wssPath** – The path of the website section from which matching content items should be returned.
- **wssObj** – The website section from which matching content items should be returned.
- **level** – The level of a category in the category hierarchy (where top-level categories have a value of 1, sub-categories of that top-level have a value of 2, etc) for which matching content items should be returned.
- **catID** – The ID of a particular category within the specified set for whose direct child categories [only] matching content items should be returned.

When omitted, the publisher defaults to the website section of the item or page currently being published.

The categoryHistogram utility will return a result for each category in the specified category set for which categorized content is found. This is a fast and efficient way of creating a histogram for the category set.

The following tags are provided to get information about the categories:

`$result.name` – returns the name of the category

`$result.count` – returns the number of currently live content items of this category targeted to the specified website section

For example, the following code:

```
<table border=1>
  #set($position=0)
  #set($histogram = $cms.newCategoryHistogram("Food Type"))
  $histogram.addWSS("/foods")
  #foreach($result in $histogram.execute())
    #set($catname=$result.name)
    #if($catname  != "Uncategorized")
      #set($nitems=$result.count)
      #if($nitems > 0 )
        #if($position==0)

<tr valign=top>
<td width="50%" border="1" align="left">
        #else
<td width="50%" border="1" align="left">
        #end
        $catname ($nitems)
        #if($position==0)
</td>
        #set($position=1)
        #else
</td></tr>
          #set($position=0)
        #end
      #end
    #end

  #end
  #if($position==1)
<td> </td>
</tr>
  #end
</table>
```

would produce output such as this:

Fruits (10)	Desserts (3)
Casserole (5)	Salad (6)
Appetizer (7)	Soup (3)
Cheese (12)	Breakfast (6)
Fruit (10)	

Optionally, a parent category can be specified (by the category's ID) at which to begin the histogram, such that only content categorized at or beneath that category (i.e. including its subcategories) will be returned.

Given the following Category Set, for example – and assuming the current section had at least one content item of each category:

Region:
- East (ID #1)
  - North East (ID #2)
    - New England (ID #3)
- West (ID #4)
  - Pacific Northwest (ID #5)
  - Calif (ID #6)

```
#set($histogram = $cms.newCategoryHistogram("Region"))
$histogram.setParentID(1)
$histogram.execute()
```

would ONLY return “North East” (again only if there was content categorizd as “North East”) – but NOT “New England” (since that is a grandchild, not direct child).

Alternately*, if you need to retrieve only categories at a certain level of the indicated category set’s hierarchy, you can use the setParentID() method, like this:

Given the following Category Set, for example – and again assuming the current section had at least one content item of each category:

Region:
- East (ID #1)
  - North East (ID #2)
    - New England (ID #3)
- West (ID #4)
  - Pacific Northwest (ID #5)
  - Calif (ID #6)

```
#set($histogram = $cms.newCategoryHistogram("Region"))
$histogram.setLevel(2)
```

would **ONLY** return histogram results for the “North East”, “Pacific Northwest”, and “Calif” categories (again, assuming there was categorized content in the current section for each of those categories).
* **Note**: setLevel() and setParentID() should not be used together, since the level explicitly stated by setLevel could conflict with the level of the children of the category specified by setParentID. Only use one or the other.

CategorySet Object

CategorySet Object	**$cms.getCategorySet**(string *setName*|int *setID*)
Given a CategorySet object $categorySet:	
String	$categorySet.name
Int	$categorySet.id
Int	$categorySet.maxLevel
Int	$categorySet.count
Category Object	$categorySet.root
Boolean	$categorySet.allowsMultiple()
Category Object	**$categorySet.getCategory**(string *categoryPathName*|int *categoryID*)
Category Object	$categorySet.getCategoryOrAll(int *categoryID*)
ArrayList of Category Objects	$categorySet.categories
ArrayList of Category Objects	$categorySet.getCategories(int *level*)
ArrayList of Category Objects	**$categorySet.getCategoryBranch**(int *parentID,* boolean *excludeParent*)

Usage:

- **setName** – the name of the category set of interest
- **setID** – the ID of the category set of interest
- **categoryPathName** – the full path name of the category (or sub-category) of interest. example: “Country/United States/California”
- **categoryID** – the ID of the category of interest
- **level** – the level of categories you want returned (where 0 = root “Uncategorized” level, 1 = main level, 2 = child-level, etc.)
- **parentID** – the ID of the category node to be, together with its subcategories, returned
- **excludeParent** – true if you want only the subcategories of the specified category to be returned, and not the category itself

Defaults:

- **excludeParent** is false by default

Category Object

Given a Category object $category:

String	$category.name
Int	$category.level
Int	$category.id
Boolean	$category.isRoot()
CategorySet Object	$category.categorySet
Category Object	$category.parent
String	$category.categorizationString
Boolean	$category.hasChildren
ArrayList of Ints	$category.displayLevels

Media

Media Items

MediaObject	**$cms.media**(int *mediaID*)
MediaObject	**$cms.media**(string *file-name*, string *media-type*)

Given a Media object $mediaItem:

Object	**$mediaItem.**attribute-name

Usage:

- **mediaID**– The specific mediaID of the media item to be used.
- **file-name**– The specific filename of the media item to be used.
- **media-type**– The media type (repository) of the media item to be retrieved, which must be one of these values:

From the Publish > Media menu

- images
- audio
- video
- documents
- binary

From the Design > Design Media menu

- designimages
- designaudio
- designvideo

attribute-name – The name of the media attribute to be output

System (non-editable) attributes:

- id (or ID)
- createDate
- modifiedDate (or lastModified)
- url
- filename
- extension
- size (bytes as a long)
- typeName

Attributes for all media:

- title
- name (the tagname)
- author
- caption

- shortCaption
- metadata
- credit
- description
- copyright
- altText
- mediaDate
- URL
- filename
- extension
- startDate
- endDate
- isLive()
- isPreview()
- isDead()
- requiresSubscription()
- subscriptionTypeIDs

Additional Attributes for images only:

- height
- width

Additional Attributes for Audio and Video only:

- runTime

Constructed Attributes (for images only):

- $util.tag(MediaObject OR MediaPlacement)
- $util.resizeImageTag(MediaObject OR MediaPlacement)

Note: MediaObjects represent the media assets as stored in the media repository. This differs from MediaPlacement objects (see the "Content" chapter), which represent a specific instance of a media repository asset for usage on a particular WCM element (e.g., a content item, website section, newsletter, etc). Thus, the attribute value of a given MediaObject (e.g., caption) will reflect the settings of the repository item, whereas the same attribute might yield a different value when retrieved off a MediaPlacement object (e.g., for a media placement that defined its own caption value). Additionally, categorization settings (explained in the "Categories" chapter) can only be retrieved off of MediaOjbects, not MediaPlacements.

Because the MediaPlacement id will always be the same as its corresponding MediaObject's id, you can always get from the MediaPlacement object to the MediaObject simply by using $cms.media([theMediaPlacement].id).

$cms.media.url vs. $cms.media($content.media.id).url

Using $cms.media.url (on a media placement object, not a media object) is unreliable with respect to the media domain that will be used. It will grab the default media domain and use that. When using $cms.media($content.media.id).url, it has a complete different routine it uses to get the URL, and will always use the correct media domain in the media URL because it has the correct context of the proper media domain. *Best Practice is to always use $cms.media($content.media.id).url and never use $mediaPlacement.url.*

Media Tag Examples:

```
$cms.media(12345).caption
$cms.media(14).altText
$cms.media("image.jpg","images").url
$cms.media("logo.gif","designimages").url
$cms.media("oursong.wav", "audio").title
$cms.media("elephantwalk.mpeg","video").size
```

The $util.formatAsBytes tag can be particularly helpful in conjunction with the $cms.media.size tag. See the "Numbers and Math" chapter for details on using this tag.

Image Tag Utility

String	$util.tag(MediaObject or MediaPlacement media)
String	$util.tag(MediaObject or MediaPlacement media, Boolean showAttributes, String extraInfo)
String	**$util.tag**(MediaObject or MediaPlacement *media*)
String	**$util.tag**(MediaObject or MediaPlacement *media*, Boolean *showAttributes,* String *extraInfo*)
String	**$util.unlinkedTag**(MediaObject or MediaDesc media)
String	**$util.unlinkedTag**(MediaObject or MediaDesc media, Boolean *showAttributes,* String *extraInfo*)

Usage:

- **media** – Is a media object, obtained through a $cms.content.media-placement tag or other means.
- **extraInfo** – Refers to additional image tag attributes to be included in <img> tag.
- **showAttributes** – indicates if height, width, border, and alt attributes should be automatically included in the <img> tag. Defaults to true.

Tag Examples:

$util.tag is a shortcut method for adding a complete HTML image tag for an image media asset (either the MediaDesc object returned by a media placement reference, or a MediaObject object returned by using $cms.media to reference the media repository directly).

Media Repository Example:
Using,

```
$util.tag($cms.media("logo.gif","designimages"))
```

to output a design image with dimensions 100 X 100 and alttext "Logo" would output:

```
<img src="http://images.website.com/designimages/logo.gif" height="100"
width="100" alttext="Logo" border=0>
```

Other Examples:

```
$util.tag($cms.media(12))
$util.tag($cms.media("logo.gif","designimages"))
```

Media Placement Example:
The tag works the same way for media placement assets, with the added feature that for a media placement that has a TargetURL value, the $util.tag outputs a properly *hyperlinked* <img> tag! So, using it on a media placement with a targetURL of "http://www.google.com" would result in:

```
<a href="http://www.google.com"><img
src="http://images.website.com/images/filename.gif" height="100" width="100"
alttext="This is alttext" border=0></a>
```

If however, you do not want the <a href> tags around your image to be automatically output (for instance, if you need to put a style class inside the <a> or otherwise customize that tag), use the $util.unlinkedTag tag instead.

Note: If the indicated media asset is not found, these tags output nothing.

Additional Options

By default, the height, width, alttext and border attributes are included in the image tag. This can be overridden as follows:

```
$util.tag($cms.content.TopImage, false)
```

which would result in:

```
<img src="http://images.website.com/images/filename.gif">
```

Additional image tag elements can also be passed into the $util.tag, like this:

```
$util.tag($cms.content.TopImage, 'align="left"')
```

which would result in:

```
<img src="http://images.website.com/images/filename.gif" height="100"
width="100" alttext="This is alttext" border=0 align="left">
```

These two options can be combined as follows:

```
$util.tag($cms.content.TopImage, false, 'align="left"')
```

which would result in:

```
<img src="http://images.website.com/images/filename.gif" align="left">
```

Note: The extraInfo param must use alternating quotes. Further, the outermost quotes must be DOUBLE quotes if you want enclosed Velocity expressions to be translated.

Resizing Image Tag Utility

Scaling Factor	
String	**$util.resizeImageTag**(MediaObject or MediaDesc *media*, int *percent*)
String	**$util.resizeImageTag**(MediaObject or MediaDesc *media*, int *percent*, boolean *showAttributes,* String *extraInfo*)
Specific Size	
String	**$util.resizeImageTag**(MediaObject or MediaPlacement *media*, int *width,* int *height*)
String	**$util.resizeImageTag**(MediaObject or MediaPlacement *media*, int *width,* int *height*, boolean *showAttributes,* String *extraInfo*)
Scaling Factor	
String	**$util.unlinkedResizeImageTag**(MediaObject or MediaPlacement *media*, int *percent*)
String	**$util.unlinkedResizeImageTag**(MediaObject or MediaPlacement *media*, int *percent*, boolean *showAttributes,* String *extraInfo*)
Specific Size	
String	**$util.unlinkedResizeImageTag**(MediaObject or MediaPlacement *media*, int *width,* int *height*)
String	**$util.unlinkedResizeImageTag**(MediaObject or MediaPlacement *media*, int *width,* int *height*, boolean *showAttributes,* String *extraInfo*)

Usage:

- **media** – a jpg, gif or png media object, obtained through a $cms.content.media-placement tag or other means.
- **percent** -scaling factor to use in adjusting image size.
- **width** – the new width of the image to be resized
- **height** – the new height of the image to be resized
- **extraInfo** –additional attributes to be included in <img> tag.
- **showAttributes** – True includes Height, Width, Border, and Alt attributes in <img> tag.

Resizing images with this tag makes the media publisher resize the image files on the fly during the publishing process. The file itself is altered to the specified height and width, adjusting the file size in the process. This is not to be confused with specifying height and width attributes in an <img> tag, which keeps the image at its original size, but simply forces the browser to load it into a space of the indicated dimensions.

Other than resizing, this tag works identically to the $util.tag described above. That means if used with a media placement that has a TargetURL, it will render the complete hyperlinked image (that is, complete with proper <a href> tags around the image). If you do *not* want the tag to automatically generate the link, then use the $util.unlinkedResizeImageTag method instead.

Notes

- Because this tag actually creates a new image file, the *first time* that it's requested, it may take a bit longer than usual to display. After that however, the resized image will already exist and so will be served with normal speed.
- The resize tag **DOES NOT** work for BMP files. If you try to resize a .bmp, it will display the image in its original size (i.e., will behave like a $util.tag).

There are two mechanisms for rescaling an image, specifying a scaling factor, or specifying new dimensions. **For example,** given an original image of dimensions of dimensions width=150, height=300…

Scaling Factor Example

Using,

```
$util.resizeImageTag($cms.content.TopImage, 50)
```

will result in:

```
<img src="http://images.website.com/images/75*150/filename.gif" width="75"
height="150"  alt="This is the alttext" border=0>
```

The image will be scaled 50% in both height and width.

Specify Dimensions Example

Using,

```
$util.resizeImageTag($cms.content.TopImage, 25, 50)
```

will result in:

```
<img src="http://images.website.com/images/25*50/filename.gif" width="25"
height="50" alt="This is the alttext" border=0>
```

The image will be scaled to a size of 25 pixels by 50 pixles. In this case, the image proportions are not maintained.

To maintain image proportions when scaling to a specific width or hight, simply set the other dimension to 0. **For example:**

```
$util.resizeImageTag($cms.content.TopImage, 50, 0)
```

will set the width to 50 and then calculate the corresponding height of the image to be 100:

```
<img src="http://images.website.com/images/50*100/filename.gif" width="50"
height="100" alt="This is the alttext" border=0>
```

The following:

```
$util.resizeImageTag($cms.content.TopImage, 0, 200)
```

will set the height to 200 and calculate the corresponding width to 100:

```
<img src="http://images.website.com/images/200*200/filename.gif" width="100"
height="200" alt="This is the alttext" border=0>
```

As with $util.tag, if the indicated media asset is not found, this tag will not output anything.

Retrieving Secure URLs

String	$util.getURL(*MediaObj*)

Usage:

- **MediaObj** – the object returned by retrieving a Media Object from a media repository, by $cms.media()

On webpages accessed via our secure server (e.g. for subscription pages restricted to paid registration users), media URLs need to be secure (i.e. https).

Images rendered via $util.tag()will be automatically output using the proper secure https URL.

However, directly accessing the url property of a MediaObject, like this:

```
$cms.media($content.image.id, "images")
```

WILL NOT give you the desired secure url.

To directly obtain the secure URL for a media asset, you must use the $util.getURL method. Examples:

```
$util.getURL($cms.media($content.image.ID), "images")
$util.getURL($cms.media("video.swf", "designvideo"))
```

Note: When accessed from a non-secure server, this tag will still output the normal, non-secure Clickability Platform URL. It's only when accessed from a secure server (e.g. secure.clickability.com) that it will return an https-based URL.

Destinations

WebsiteSection Object	$cms.websiteSection
WebsiteSection Object	**$cms.websiteSection**(string website-section-name|website-path)
WebsiteSection Object	**$cms.websiteSection**(string domain, string website-section-name|website-path)
WebsiteSection Object	$cms.websiteSection(int *id*)
WebsiteSection Object	$cms.parent
WebsiteSection Object	**$cms.parent**(string website-section-name|website-path)
WebsiteSection Object	**$cms.parent**(string domain, string website-section-name|website-path)
WebsiteSection Object	**$cms.parent**(WebsiteSectionObject *wss*)

Given a Website Section object $wss:

Object	**$wss.**attribute-name
ArrayList	$wss.children
Object	$cms.hierarchyField(*attribute-name*)
MediaObject	$cms.hierarchyMedia(*attribute-name*)

Usage:

- **Website-section-name** – The name of the website section item to be used. When omitted, the publisher defaults to the website section of the item or page currently being published.
- **website-path** – The full path to the website section page. The value specified must start with a forward slash ('/'). To specify the domain's homepage, use a single forward slash. When omitted, the publisher defaults to the website section of the item or page currently being published.
- **attribute-name** – The name of the website section attribute to be displayed

Default Attributes (for website sections):

- id
- name
- description
- pathSegment
- path
- destinationPath (for use on sites with 'Strict Hub Page URL' enabled)
- publishRSS
- typeName

System Attributes (for website sections):

- url
- children
- modifiedDate
- createDate

The websiteSection tag retrieves the current website section or the specified website section. The parent tag retrieves either the parent of the current website section or the *parent* of the specified website section. The parent and website section tags are identical in all other ways.

Examples:

```
$cms.websiteSection.name
$cms.websiteSection("/home").description
$cms.parent.name
$cms.parent(27).description
```

Additionally, the publishing system provides the URL of a website section:

```
$cms.websiteSection.url
$cms.websiteSection("/news").url
```

It also provides the path to that section, as well as the path segment (i.e., directory):
For example, for the section at http://www.clickability.com/products/wma:

```
$cms.websiteSection.path
```

- would return “/products/wma”.

```
$cms.parent.path
```

- would return “/products”.

```
$cms.websiteSection("wma").pathSegment
```

- would return “wma”.

The publishRSS attribute is a boolean indicating if the website section is configured to publish the standard [checkboxed] RSS feeds (true) or not (false).

Hierarchy Tags

The $cms.hierarchyField tag allows you to traverse up the destination hierarchy looking for an attribute value. It will start at the current Destination, working up the destination hierarchy until it finds a Destination with a non-null value for that attribute.

If no non-null value is found for that attribute by the time it reachest the topmost, domain-level Destination, the tag will return null.

Examples:

Keywords Example:
Given the following Destinations with the following values for a “metaKeywords” attribute:

Destination:		metaKeywords attribute value
www.site.com		“technology, distributed, company”
News		“press release, merger”
	Local	attribute value blank
Information		attribute not defined
	Contact	attribute value blank

A $cms.hierarchyField("metaKeywords") tag encountered on a page located in the “Contact” section would return the string: “technology, distributed, company”.

That same tag encountered on a page located in the “Local” section would return the string “press release, merger”.

Similarly, the $cms.hierarchyMedia tag allows you to traverse up the destination hierarchy looking for a media placement value.

Media Example:
Given the following Destinations with the following values for a “headerImage” media placement:

Destination:		headerImage value
www.site.com		generic_siteHeader.jpg
News		news_header.jpg
	Local	attribute value blank

A $cms.hierarchyMedia("headerImage") tag encountered on a page located in the “Local” section would return the news_Header.jpg MediaObject.

Newsletters (v2)

Newsletter Elements

Newsletter Object $cms.newsletter

Given a Newsletter Object $nl:

Object **$nl.**attribute-name

Usage:

- Allows access to the fields of the newsletter (Automated Mailing) being sent.
- **attribute-name** – The name of the newsletter attribute to be output (derived from the newsletter-type definitions) or a system attribute

Default Attributes (for newsletters):

- name
- description

System Attributes (for newsletters):

- id
- modifiedDate
- createDate
- typeName
- typeID
- typeVersion

Example:

```
Welcome to $cms.newsletter.name!
```

Mailing Elements

Mailing Object $cms.mailing

Given a Mailing object $myMailing:

Object **$myMailing.**attribute-name

Usage:

- Standard Mailings (not Automated) only: allows access to the fields of the Mailing item being published.
- **attribute-name** – The name of the Mailing attribute to be output (derived from the mailing-type definitions) or a system attribute

Default Attributes (for mailings):

- name
- description

System Attributes (for mailings):

- id
- modifiedDate
- createDate
- typeName – the mailing-type name
- typeID – the mailing-type id
- typeVersion – the mailing-type version

Example

```
This week's top news: $cms.mailing.newsField
```

Mailing List Elements

MailingList Object	$cms.mailinglist
MailingList Object	**$mailinglist** (auto-responders only)

Given a MailingList object $ml:

Object	**$ml.**attribute-name

Usage:

- Allows access to the fields of the Mailing List to which the newsletter is being sent.
- **attribute-name** – The name of the Mailing List attribute to be output

Attributes:

- mailinglistid
- domain
- name
- ownerAddress
- listemailname
- listemail
- replyemail
- listname

Note: In auto-responder templates (published from the mailing server), the above tags can be used without the $cms prefix: e.g., $mailinglist.name.

Examples:

In the newsletter template itself:

```
To provide feedback on our newsletter, email ${cms.mailinglist.replyemail}.
```

In mailing list auto-responder template:

```
Thank you for subscribing to ${mailinglist.name}.
```

Mailing List Subscription Elements

To specify text for Mailing List autoresponse emails, create a single master template with conditions for each type of email message, and assign that template on the Subscription tab of the Mailing List. If a condition is missing from the master template, the system will insert the default text below:

Template Condition	Default Text	Function
isSubscribeResponse	You have been subscribed to the mailing list.	User is successfully subscribed to list
isSubscribeConfirmationRequest	Reply to this email to complete subscription to mailing list name.	User needs to reply to this before being subscriber
isSubscribeError	Unable to subscribe sender to the mailing list.	User is not subscribed
isUnsubscribeResponse	You have been unsubscribed from the mailing list.	User has successfully unsubscribed from the mailing list
isUnsubscribeConfirmationRequest	Reply to this email to complete unsubscription from mailing list name.	User needs to reply to this email to complete unsubscription
isUnsubscribeError	Unable to unsubscribe sender from the mailing list	User was not unsubscribed

Example:

```
#if($isSubscribeResponse)
Congratulations! You will now be receiving the $mailinglist.name newsletter
in your inbox. To unsubscribe, you can do so at any time by sending us an
email at ${mailinglist.listemailname}-unsubscribe@nl.clickability.com and
entering "Unsubscribe" into the subject line.
#elseif($isUnsubscribeResponse)
We have removed your email address $subscriber.email from our
$mailinglist.name Newsletter subscriber list.
#elseif…
```

Refer to the Clickability Platform User Community at **http://community.clickability.com** for a complete Mailing List Autoresponder template.

Subscriber Elements

Subscriber Object	$cms.subscriber
Given a Subscriber object $mySubscriber:	
Object	**$mySubscriber.***attribute-name*
Boolean	$mySubscriber.subscribedTo(int *mailingListId*)
Boolean	$mySubscriber.pendingSubcriptionTo(int *mailingListId*)
Date	$mySubscriber.dateSubscribed(int *mailingListId*)

Date	$mySubscriber.dateSubscribeConfirmed(int *mailingListId*)

Usage:

- Can only be used in mailing-list un/subscribe auto-responder templates, subscription-protected pages of the site (like edit profile pages), or in newsletter templates (Standard or Automated) that have Personalization enabled. Allows access to the fields of the subscriber to whom the newsletter is being sent.
- **mailingListID** – the id of the mailing list of interest

Attributes:

- ID
- username
- email
- firstname
- lastname
- birthdate
- title
- company
- address1
- address2
- city
- state
- zipcode
- country
- custom1
- custom2
- custom3
- custom4
- custom5
- custom6
- custom7
- custom8
- custom9

Newsletter Personalization using Subscriber data

Using Conditionals & Variables to personalize Newsletter templates:

*__Note__: Personalization code will ONLY work if your account has been set up for Personalization AND the Newsletter has the 'personalization' box checked.

Personalization means that after all other Velocity in your newsletter templates has been translated, the mailserver will do a second-pass to replace all $subscriber expressions with their corresponding values (upon which it can then perform only very limitied, string-based operations).

For this reason, to use conditionals related to the subscriber fields, those conditional template tags must pass intact from the newsletter generation to the mailserver to be translated – therefore they must be escaped during the initial translation:

```
\#if($subscriber.city == "San Francisco")
You are local.
\#else
You live elsewhere.
\#end
```

or

```
\#set($foo = $subscriber.city)
```

Note: Escaping **IS NOT** necessary in mailing list un/subscribe auto-responder templates since they are only invoked from the mailserver.

Again, since the second pass translation that happens on the mailserver is really only equipped to handle the $subscriber object and basic string functions (i.e., it does NOT have access to other libraries such as $cms, $util, $math, etc.) you will be limited in what operations may be done with the subscriber values once they are inserted.

Special Newsletter Usage of Standard Site Publishing Tags

Content

```
$cms.content(11,"/worldnews").url
$cms.content(11,false).contentTargets
```

Note: Since there is no concept of "current website Section" in a newsletter, $cms.content(id) will never work. Instead you must either indicate the section, or use the $cms.content(id,false) method to retrieve the object, then look up its contentTargets.

HTML Email View Tracker

Outputs a no javascript tracker:

```
$imware.tracker
```

Note: <noscript> version is only used when newsletter is actually mailed. In newsletter preview source code, you will still see the <script> version.

Sent Mailings

SentMailing Object	$cms.sentMailing(int *mailingID*)
Given a SentMailing object $myMailing:	
Object	**$myMailing.**attribute-name
ArrayList of SentMailing Objects	**$cms.sentMailings**(int *newsletterID* [,Date *fromDate*] [,Date *toDate*] [,int *numberReturned*])

Usage:

Sent Mailings are constructed from the newsletter email itself, not from the setup objects that defined that

newsletter. They therefore only provide access to parts of the actual email that was sent for the specified mailing.

- **attribute-name** – The name of the Sent Mailing attribute to be output
- **fromDate** – the earliest date you want to retrieve mailings for
- **toDate** – the latest date you want to retrieve mailings for
- **numberReturned** – the number of Sent Mailing results to return

System Attributes (for mailings):

- **mailingID** – the id of the Mailing object
- **mailingListID** – the id of the list to which the mailing was sent
- **mailngListName** – the name of the list to which the mailing was sent
- **newsletterID** – the id of the newsletter for which the mailing was sent
- **sentDate** – the date the mailing was sent
- **subject** – the Subject line of the email that was sent
- **html** – the html body content of the mailing
- **text** – the text body content of the mailing
- **url** – the url at which the mailing can be viewed online (html version)
- **htmlurl** – the url at which the html version of the mailing can be viewed online
- **texturl** – the url at which the text version of the mailing can be viewed online
- **newsletterID** – the id of the Newsletter mailing you want to access

When requesting an Array of Sent Mailing objects, the Sent Mailing results are returned in order by date sent, most recent first.

Examples:

Get a specific mailing by its mailing id:

```
#set($myMailing = $cms.sentMailing(1234))
```

Iterate the last 20 mailings sent for newsletter id 37:

```
#foreach($myMailing in $cms.sentMailings(37,20))
```

Get the last 20 mailings sent for newsletter 37 between given dates:

```
#set($date1=$util.parseDate("09/15/04","MM/dd/yy"))
#set($date2=$util.parseDate("11/23/04","MM/dd/yy"))
#set($mailings=$cms.sentMailings(37,$date1,$date2,20))
```

To display a linked archive of all past issues of a given newsletter, 54, mailed out to a particular Mailing List, “NL Subscribers”:

```
#foreach($sentNL in $cms.sentMailings(54))
	#if($sentNL.mailingListName == "NL Subscribers")
		<a href="$sentNL.url">$sentNL.subject</a>
	#end
#end
```

Publishing Information

String	$cms.domain
String	$cms.mediaDomain
String	$cms.path
String	$cms.destinationPath
String	$cms.envHost
Boolean	$cms.isDev
Boolean	$cms.isStage
Boolean	$cms.isProduction
Boolean	$cms.isContentPreview
Boolean	$cms.isPlacementListPreview
Content Object	$cms.content
WebsiteSection Object	$cms.websiteSection
ArrayList of TemplateObjects	$cms.templateStack

These tags return information from the publisher regarding the current publishing task.

If the page requested is http://www.domain.com/news/health/12345.html then,

```
$cms.domain
$cms.path
$cms.destinationPath
```

would display:

- www.domain.com
- /news/health
- /news/health/

Note: the presence of the leading slash.

$cms.path results in paths that do not include trailing slashes for destinations with “Strict Hub Page Handling” enabled at the domain level. This can cause 301 redirects to be generated for URLs built with $cms.path.

To retrieve the URL with trailing slashes, use $cms.destinationPath.

The $cms.envHost is helpful when you need to manually construct a full URL in a page, but want to make sure the reference stays within the same publishing environment (development, staging, or production).
The tag returns the following results:

Publishing Environment	Output
Development	dev.
Staging	stage.
Production	

Because a period is returned as part of the results for development and staging environments, this tag is best used as follows:

```
<a href="http://${cms.envHost}www.mydomain.com">
```

When this code is published in the different environments, it will produce:

in development: `<a href="http://dev.www.mydomain.com">`
in staging: `<a href="http://stage.www.mydomain.com">`
in production: `<a href="http://www.mydomain.com">`

The `$cms.isContentPreview` tag is helpful if you want the template assigned for a given content type to do one thing when used in preview mode (e.g. display the whole content item as a single page), but a different thing when actually publishing the site (e.g. show only one page at a time, as requested in the URL). This tag will only return true when you preview a page via the "Preview" button from within a Content Item.

Similarly, the `$cms.isPlacementListPreview` tag is helpful if you want the template for a given section to do one thing when Previewing from the Placement List screen, but a different thing when actually publishing the site. It will only return true when you preview a page via the "Preview" button on a Placement List.

The `$cms.content` and `$cms.websiteSection` tags return the content item being published and the website section being published, respectively. If a hub page is being published, there is no content item and $cms.content will return null.

The `$cms.templateStack` tag can be used to get the name (and other properties) of the current template - which would be last in the array - but also of the topmost calling template - which would be at index 0.

ArrayList of TemplateObjects	$cms.templateStack
Given a Template object $myTemplate	
Object	$myTemplate.attribute-name
Attributes:	
String	$myTemplate.name
String	$myTemplate.getTypeName
Int	$myTemplate.ID
Int	$myTemplate.folderID
String	$myTemplate.comments
Int	$myTemplate.version
Boolean	$myTemplate.isLatestVersion
String	$myTemplate.folderName
String	$myTemplate.filename
String	$myTemplate.description
Boolean	$myTemplate.contentType
Date	$myTemplate.createDate
Date	$myTemplate.lastModified
String	$myTemplate.body

Request Information

URL Information

String	$req.fullURL
String	$req.url
String	$req.fullURI
String	$req.uri
String	$req.queryString

Note:
See Section II below for more functions for working with query strings and request parameters.

WARNING:
These tags do not produce the expected results when previewing templates. They will have null values as the request object is only fully constructed by the Clickability Platform publishers, including all three of the Development, Staging and Production publishing environments.

These tags allow you to get and utilize the entire requested URL, or just a portion of it.

If this publisher is publishing a page requested by this URL:

http://www.fairoaksgazette.com/sports?abc=123&x=z

The above tags produce the following results.

Tag	Output
$req.fullURL	http://www.fairoaksgazette.com/sports?abc=123&x=z
$req.url	http://www.fairoaksgazette.com/sports
$req.fullURI	/sports?abc=123&x=z
$req.uri	/sports
$req.queryString	abc=123&x=z

Request Parameters

String	**$req.param**(string *paramname*)
Integer	**$req.intParam**(string *paramname*)
ArrayList	**$req.paramValues**(string *paramname*)
ArrayList	$req.params
String	**$util.addGetParameter**(string *url*, string *paramname*, string *paramvalue*)

Usage:

- **paramname** – The the name of a parameter passed in the query string portion of the URL
- **parmvalue** – The value of the querystring parameter

Usage

To retrieve a single-valued parameter as a string:

```
$req.param("paramname")
```

To retrieve a single-valued parameter as an integer:

```
$req.intParam("paramname")
```

To retrieve all values of a multi-valued parameter:

```
#foreach($paramvalue in $req.paramValues("paramname"))
	$paramvalue
#end
```

To retrieve all the parameters:

```
#foreach($paramname in $req.params)
	$req.param($paramname)
#end
```

To append name=value request parameter pairs to a URL:

Given a $url value of http://www.site.com/news/1234.html
and the following line of code:

```
#set($adjustedURL = $util.addGetParameter($url, "page", "2"))
```

$adjustedURL would return:

http://www.site.com/news/1234.html?page=2

Polls

PollObject	**$cms.poll**(int *pollid*)

Given a Poll object `$poll`:

Integer	**$poll.votes**(int *answer-id*)
Integer	$poll.totalVotes
Integer	$poll.totalResponses

Usage:

- **pollid -** corresponds to the contentID of the content item representing a poll
- **answer-id -** an integer representing the index id of the answer

Using Polls

This loads the top ranked item from the Polls website section and places it into the object $mypoll:

```
#set($mypoll= $cms.poll($cms.link("Polls",1).id))
```

To retrieve the results for a particular answer item, use the tag:

```
$mypoll.votes(n)
```

where **n** is the index id of the answer (usually used in conjunction with a Loop).

The following tags:

```
$mypoll.totalVotes
$mypoll.totalResponses
```

return respectively:

the total number of votes
the total number of responses

All of the $poll tags return integer values so that they can be used in equations. For example, to calculate percentages:

```
$percentage= $mypoll.votes(n)/$mypoll.totalReponses * 100
```

Example

The following code will generate a horizontal bar graph of the poll results for the Poll content item being published:

```
#set($poll=$cms.poll($cms.content.id))
Poll Results
$cms.content.title
<table>
#foreach($choice in $cms.content.answers)
  #set($colorimg="red.gif")
  #if(($velocityCount % 2) == 1)
    #set($colorimg="gray.gif")
  #end
  #set($percent=$poll.votes($velocityCount)*100 /$poll.totalResponses )
  <tr>
  <td width=150>
       $choice<br/>${percent}%
  </td>
  <td align="left" width=337>
  <img src="$cms.media('$colorimg', 'designimages').url" height=25
width="${percent}%">
  </td>
  </tr>
#end
</table>
```

Surveys (version 1)

The older “version 1” version of Surveys has been deprecated and will no longer be supported. See Surveys (version 2) for details about the current Survey implementation.

Surveys (version 2)

Producing an HTML Survey Form

Setting up a version 2 survey requires creating a content type that contains one (and ONLY one) field of type Survey, and creating a Survey template to output that survey's contents, using a couple of simple template tags.

The Survey field contains a Title, Header, Footer, and any number of custom-defined questions in between.

The $cms.showSurvey() tag automatically generates the entire survey HTML code.

String	**$cms.showSurvey**(ContentObject *$content*, SurveyField *$content.survey*)
String	**$cms.showSurvey**(ContentObject *$content*, SurveyField *$content.survey*, int n*)*
String	**$cms.showSurvey** (ContentObject *$content*, SurveyField *$content.survey*, int *n*, String *emailaddress*, string rurl*)*

Usage:

- **$content** – a content item containing the Survey field, obtained by $cms.content or other means.
- **$content.survey** – the tagname of the content item's survey field
- **n** – the maximum number of responses per user (cookie based) that will be counted
- **emailaddress** – an address to which each survey entry will be emailed
- **rurl** – the [absolute] url to which the user should be taken after the survey is processed

Defaults:

- **n** – If none specified, defaults to 1
- **rurl** – If no rurl specified, the user will be taken back to the survey form after submitting their results

Applying Styles to the HTML Survey Form

String	$cms.showSurveyCSS()
String	$cms.showSurveyCSS(String *theme)*
String	**$cms.showSurveyCSS**(String *colors [*,String *fontSizes])*

Usage:

- **theme**– specifies the use of 1 of 3 pre-defined stylesheets other than the default (available themes: lilac, gray/white, banana).
- **colors** – a comma-separated list of the following format:
  - "questionBG, responseBG, borderBG [, questionFG [, responseFG]]"

where:

- **questionBG** – a 6-character hexadecimal code to set the background color of the survey question fields
- **responseBG** – a 6-character hexadecimal code to set the background color of the survey response fields
- **borderBG** – a 6-character hexadecimal code to set the background color of the field borders

- **questionFG** – a 6-character hexadecimal code to set the foreground color of the survey question fields
- **responseFG** – a 6-character hexadecimal code to set the foreground color of the survey response fields
- **fontSizes** – a comma-separated list of the following format:
    - "normalSize [, largeSize [, xlargeSize]]"

where:

- **normalSize** – the size style assignment for normal survey text elements
- **largeSize** – the size style assignment for the larger survey text elements
- **xlargeSize** – the size style assignment for the largest survey text elements
- **emailaddress** – an address to which each survey form submission will be emailed
- **rurl** – the url to which the user will be redirected after submitting the survey

The survey HTML output by the $cms.showSurvey() tag contains fixed style class assignments. The $cms.showSurveyCSS() tags allow you to control the look and feel of the survey output by customizing the stylesheet to which these style class assignments refer.

Using the Default Stylesheet

The basic $cms.showSurveyCSS() tag will output the following stylesheet:

```
<style type="text/css">
table.cmP_survey {width: 100%; }
table.cmP_survey table { background-color: #eeeeee; }
table.cmP_survey td.cmP_vBorder {width: 1px; background-color: #888888}
table.cmP_survey td.cmP_veBorder {width: 1px; background-color: #888888}
table.cmP_survey td.cmP_hBorder {height: 1px; background-color: #888888}
table.cmP_survey td.cmP_heBorder {height: 1px; background-color: #888888}
table.cmP_survey td.cmP_title {padding-left: 15px; padding-top: 4px;
padding-bottom: 4px; font-weight: bold; font-size: 133%; background-color:
#cccccc; }
table.cmP_survey td.cmP_header {padding-left: 25px; padding-top: 8px;
padding-bottom: 8px; background-color: #eeeeee; }
table.cmP_survey td.cmP_submit {padding-left: 15px; padding-top: 4px;
padding-bottom: 4px; font-weight: bold; font-size: 133%; background-color:
#cccccc; }
table.cmP_survey td.cmP_footer {padding-left: 25px; padding-top: 8px;
padding-bottom: 8px; background-color: #eeeeee; }
table.cmP_survey td.cmP_question {padding-left: 10px; padding-top: 4px;
padding-bottom: 4px; font-weight: bold;  background-color: #cccccc; }
table.cmP_survey td.cmP_responses {padding-left: 25px; padding-top: 8px;
padding-bottom: 8px; background-color: #eeeeee; }
table.cmP_survey td.cmP_questionSpacer {height: 30px}
table.cmP_survey td.cmP_submit a {}
table.cmP_survey span.cmP_requiredQuestion {padding-left: 4px; color: red;
font-weight: bold}
table.cmP_survey span.cmP_requiredSubQuestion {padding-left: 4px; color:
red; font-weight: bold}
table.cmP_survey span.cmP_questionNumber {padding-right: 5px}
table.cmP_edit td {text-align: center; padding-left: 5px; padding-right:
5px; padding-top: 2px; padding-bottom: 2px; font-weight: bold; color:
#00000}
table.cmP_matrix td.cmP_rowLabel {text-align: center}
table.cmP_matrix td.cmP_columnLabel {text-align: center}
table.cmP_matrix td.cmP_selector {text-align: center}
```

```
table.cmP_matrix td.cmP_hSpacer {width: 30px}
table.cmP_matrix td.cmP_vSpacer {height: 10px}
table.cmP_matrixFlipped td.cmP_rowLabel {text-align: center}
table.cmP_matrixFlipped td.cmP_columnLabel {text-align: center}
table.cmP_matrixFlipped td.cmP_selector {text-align: center}
table.cmP_matrixFlipped td.cmP_hSpacer {width: 30px}
table.cmP_matrixFlipped td.cmP_vSpacer {height: 10px}
table.cmP_matrixDropdown td.cmP_rowLabel {text-align: center}
table.cmP_matrixDropdown td.cmP_selector {text-align: center}
table.cmP_matrixDropdown td.cmP_hSpacer {width: 12px}
table.cmP_matrixDropdown td.cmP_vSpacer {height: 12px}
table.cmP_matrixDropdownFlipped td.cmP_columnLabel {text-align: center}
table.cmP_matrixDropdownFlipped td.cmP_selector {text-align: center}
table.cmP_matrixDropdownFlipped td.cmP_hSpacer {width: 24px}
table.cmP_matrixDropdownFlipped td.cmP_vSpacer {height: 8px}
table.cmP_mcDropdown td.cmP_selector {text-align: center}
table.cmP_mcHorizontalLR td.cmP_label {text-align: center}
table.cmP_mcHorizontalLR td.cmP_selector {text-align: center; padding-left:
3px; padding-right: 3px}
table.cmP_mcHorizontalLR td.cmP_spacer {width: 18px}
table.cmP_mcHorizontalLR td.cmP_spacerOther {height: 10px}
table.cmP_mcHorizontalUD td.cmP_label {text-align: center; padding-left:
6px; padding-right: 6px}
table.cmP_mcHorizontalUD td.cmP_selector {text-align: center; padding-top:
3px; padding-bottom: 3px}
table.cmP_mcHorizontalUD td.cmP_spacerOther {height: 10px}
table.cmP_mcVertical td.cmP_label {text-align: left}
table.cmP_mcVertical td.cmP_selector {text-align: center; padding-right:
6px}
table.cmP_mcVertical td.cmP_spacer {height: 8px}
table.cmP_textHorizontal td.cmP_label {text-align: center}
table.cmP_textHorizontal td.cmP_input {text-align: center; padding-top: 3px}
table.cmP_textHorizontal td.cmP_spacer {width: 20px}
table.cmP_textVertical span.cmP_label {padding-left: 15px}
table.cmP_textVertical td.cmP_spacer {height: 8px}
</style>
```

Using the Default with Custom Colors

To use the default styles, but with different colors, you may pass in the colors as hexadecimal values. The colors must be specified in the sequence *questionBG*, *responseBG*, *borderBG*, *questionFG*, *responseFG* (where the first 3 are required).

```
$cms.showSurveyCSS("cccccc,eeeeee,999999")
```

assigns a value of “cccccc” to *questionBG*, “eeeeee” to *responseBG*, and “999999” to *borderBG*.

```
$cms.showSurveyCSS("cccccc,eeeeee,999999,ffcc99")
```

assigns a value of “cccccc” to *questionBG*, “eeeeee” to *responseBG*, “999999” to *borderBG*, and “ffcc99” to *questionFG*.

Using the Default with Custom Colors AND Sizes

To use the default styles, but with different colors and font sizes, you may pass in the colors as hexadecimal values, followed by the sizes. The colors must be specified in the sequence *questionBG*, *responseBG*, *borderBG*, *questionFG*, *responseFG* (where the first 3 are required). The sizes must be specified in the sequence *normalSize*, *largeSize*, *xlargeSize* (where only the first is required).

```
$cms.showSurveyCSS("cccccc,eeeeee,999999", "12px,15pt")
```

assigns a value of “cccccc” to *questionBG*, “eeeeee” to *responseBG*, and “999999” to *borderBG*, and a style definition of 12px to normal text elements, 15pt to large-size text elements.

Using a pre-set Theme

In addition to the default, the system also offers pre-defined stylesheets or “themes”: the 3 currently available are: lilac, gray/white, or banana. To use one of these stylesheets, just pass the theme:

```
$cms.showSurveyCSS("gray/white")
```

Overriding selected styles of the $cms.showSurveyCSS() stylesheet

If you wish to override a style definition output by the $cms.showSurveyCSS() tag, you can do so by placing your own style definition (with the same class name) below the $cms.showSurveyCSS() call:

```
<head>
  $cms.showSurveyCSS()
  table.cmP_survey { width: 95%; }
</head>
```

Using your own custom stylesheet

If you don’t want to use any of the default styles, you can omit the $cms.showSurveyCSS() tag altogether, and just add your own stylesheet to your template’s <head></head> element. Of course, the style name(s) you use there will still need to correspond to the style attribute(s) of the HTML element(s) in the survey HTML.

Using your own custom HTML

If you want to use your own custom HTML, you can omit the $cms.showSurvey() tag and instead use the $cms.getSurvey to retrieve a SurveyObject to use with your own HTML.

SurveyObject	**$cms.getSurvey**(ContentObject *$content*, SurveyField *$content.survey*)
SurveyObject	**$cms.getSurvey**(ContentObject *$content*, SurveyField *$content.survey*, int n*)*
SurveyObject	**$cms.getSurvey** (ContentObject *$content*, SurveyField *$content.survey*, int *n*, String *emailaddress*, string rurl*)*

Usage:

- $**content** – a content item containing the Survey field, obtained by $cms.content or other means.
- **$content.survey** – the tagname of the content item's survey field
- **n** – the maximum number of responses per user (cookie based) that will be counted
- **emailaddress** – an address to which each survey entry will be emailed
- **rurl** – the [absolute] url to which the user should be taken after the survey is processed

Defaults:

- **n** – If none specified, defaults to 1
- **rurl** – If no rurl specified, the user will be taken back to the survey form after submitting their results

The Survey Object returned can only be used for Survey fields of type Text, Text Area and Multiple Choice. Survey fields of type Matrix or BuiltIn are not available through the Survey Object.

Given a Survey object **$mySurvey**

Object	$mySurvey.attribute-name

Attributes (for Survey objects):

- questionList - list of survey questions
- hiddenFields - the hidden fields that must be passed in order to submit the survey response
- Recaptcha – the Recaptcha script
- InvalidCaptchaMessage – the error message shown when an invalid Recaptcha is submitted

The Survey Object attributes can also be retrieved through the following methods:

- List getQuestionList()
- String getHiddenFields()
- String getRecaptcha()
- String getInvalidCaptchaMessage()

The getQuestionList() method returns a list of survey question. Each Survey Question Object has a set of attributes.

Given a Survey Question Object $myQuestion

Object $myQuestion.attribute-name

Attributes (for Survey Question objects):

- **Int questionID -** the question identifier
- **String questionText -** the text of the question
- **SurveyTestAnswerObject TextAnswer (if appropriate) -** the text of the answer for a Text question
- **List of SurveyMultipleChoiceAnswerObjects multichoice (if appropriate)** – a list of the answers for a multiple choice question

The Survey Question Object attributes can also be retrieved through the following methods:

- String getQuestion()
- SurveyQuestionTextAnswerObject getText()
- List of SurveyMultichoiceAnswerObjects getMultichoice()

Working with Survey Result Data

Survey results are retrieved in the Clickability Platform, by opening the Survey-containing content item and clicking "View Results". They can be shown as either individual or aggregate results, and can also be exported to a csv file.

When a user submits a survey, the url to which they are redirected will contain the following parameters which can be used to assess additional information about their submission.

For example, given the following tag:

```
$cms.showSurvey($content, $content.survey, 1, "dept@co.com",
"http://www.survey.com/results")
```

the user will be taken to the following url upon submitting the survey:

http://www.survey.com/results?responded=true&responses=1

Survey Response Parameters

- **responded** – boolean indicating whether or not the users vote was actually counted. Determined by enforcing max number of responses per user (based on cookie)
- **responses** – the number of responses the survey successfully counted for the user (based on their cookie). * Set to -1 if age-restriction (set as Birth Date validation in Survey setup) fails.

Working with Survey Response Aggregate Information

For surveys with multiple choice fields, it is possible to retrieve the count and percentage information for the survey responses.

List	**$cms.surveyResponses**(ContentObject *$content*, SurveyQuestionObject *$content.surveyQuestion*)

Usage:

- **$content** – a content item containing the Survey field, obtained by $cms.content or other means.
- **$content.surveyQuestion** – a Survey Question Object, obtained by $cms.content.questions or other means

The $cms.surveyResponses() method returns a list of survey response questions. Each Survey Response Question Object has a set of attributes.

Given a Survey Response Question Object $myResponse

- Object $myResponse.attribute-name

Attributes (for Survey Response Question objects):

- **Int questionID -** the question identifier
- **String question** – the text of the question
- **List of SurveyResponseAnswerObjects answers** – list of survey response answers

Each Survey Response Answer Object has a set of attributes.

Given a Survey Response Answer Object $myAnswer

- Object $myAnswer.attribute-name

Attributes (for Survey Response Answer objects):

- **String answer -** the text of the answer
- **Int count -** the count of responses with the answer
- **Int percent -** the percentage of response with the answer

Calendars

Note: The old DisplayCalendar methods (special objects for constructing/displaying calendars) have been deprecated.

In their place, Clickabiilty has released the capability to have a 'Calendar' tab on content items (configured on the Content Type 'edit settings' screen, by checking 'Enable Events').

The Calendar tab lets you assign one or more schedules for a content item – including one-time schedules (like "May 5-7, 2008") and/or complex recurring schedules (like the 1st Thursday of every month).

When content items are scheduled using the Calendar tab, the date of each occurrence (of which there would be 12 in a given year using the 1st Thursday of every month example) is search-indexed, such that we can later search very efficiently for content items that have an 'event' (i.e. Calendar tab occurrence) on a specific date or date range.

The methods needed to search for content items with occurrences on a given date range are described in the "Calendar Schedule Filters" section of the "Site Search" chapter.

The methods and attributes for simply displaying a content item's Calendar Schedule information are found in the "Calendar Schedules on Content" section of the "Content Tags" chapter.

And methods for submitting content with schedule information mapped to its Calendar Tab are described in the "Specifying Calendar Schedule Values" section of the "Content Submissions" chapter.

Interactive Tools Integration

Standard SAVE THIS, EMAIL THIS, PRINT THIS, MOST POPULAR & RSS Buttons

String	$imware.buttons
String	$imware.buttons(int *destid)*
String	**$imware.buttons**([int *destid*,] String *horiz-vert)*
String	**$imware.buttons**([int *destid*,] String *icon-color,* String *text-color)*
String	**$imware.buttons**([int *destid*,] String *icon-color,* String *text-color,* String *horiz-vert)*
String	**$imware.buttons**([int *destid*,] boolean *ST*, boolean *ET*, boolean *PT*, boolean *MP*, boolean *RF*, boolean *wrap*, String *icon-color,* String *text-color,* String *horiz-vert)*

Usage:

- **destid –** the destination ID of the section being published

Use this tag to include the standard buttons of the Interactive tool suite onto pages published by the Clickability Platform. Parameters are described below:

Tag Parameter	Tag	Values	Default
destid	Corresponds to the PID of the tools for this section of the site. Work with your Client Services Representitive on establishing these.	any number	WSS id of page being published
horiz-vert	Either Horizonal buttons or vertically stacked buttons	H or V	H
icon-color	The color of the icon in the buttons	any web safe hex color	cc0000
text-color	The color of the text in the buttons	any web safe hex color	cc0000
ST	Include SAVE THIS button?	true or false	true
ET	Include EMAIL THIS button?	true or false	true
PT	Include PRINT THIS button?	true or false	true
MP	Include MOST POPULAR THIS button?	true or false	true
RF	Include RSS FEEDS button?	true or false	false
wrap	Button text wrapped onto 2 lines?	true or false	false

Custom SAVE THIS, EMAIL THIS, PRINT THIS, MOST POPULAR & RSS Buttons

String	$imware.customButtonCode
String	$imware.customButtonCode(int *destid*)

Usage:

- **destid** – the destination ID of the section being published

Use this tag to include your own custom button images for the Interactive tool suite onto pages published by the Clickability Platform.

This will output the javascript functions to be called by the buttons, but **NOT** the code to render the actual buttons. The code for each button must be added to the template in the desired button location, and should look like this (subsituting the filepath to your custom button images as appropriate):

For SAVE THISTM:

```
<a href="#" ONCLICK="return(ST());" onMouseOver="return(STMouseOver());"
onMouseOut="return(STMouseOut());"><img src=”YOUR_ST_IMG_LOCATION_HERE”></a>
```

For EMAIL THISTM:

```
<a href="#" ONCLICK="return(ET());" onMouseOver="return(ETMouseOver());"
onMouseOut="return(ETMouseOut());"><img src=”YOUR_ET_IMG_LOCATION_HERE”></a>
```

For PRINT THISTM:

```
<a href="#" ONCLICK="return(PT());" onMouseOver="return(PTMouseOver());"
onMouseOut="return(PTMouseOut());"><img src=”YOUR_PT_IMG_LOCATION_HERE”></a>
```

For MOST POPULAR:

```
<a href="#" ONCLICK="return(MP());" onMouseOver="return(MPMouseOver());"
onMouseOut="return(MPMouseOut());"><img src=”YOUR_MP_IMG_LOCATION_HERE”></a>
```

For RSS:

```
<a href="#" ONCLICK="return(IR());" onMouseOver="return(IRMouseOver());"
onMouseOut="return(IRMouseOut());"><img
src=”YOUR_RSS_IMG_LOCATION_HERE”></a>
```

Configure Print This Page Display

The Clickability Platform Print This tool enables you to offer your readers the option to print a printer-friendly version of your webpages without extraneous text, advertisements and/or graphics. Print This achieves this by looking for specific HTML tags that identify what information you want included in the printer-friendly version as well as what information you want excluded. You can choose to include or exclude text, advertisements, navigation, graphics, etc. by utilizing the simple "include" and "exclude" HTML comment tags described below.

Tag Name	Purpose
<!--startclickprintinclude-->	Indicates the beginning of the content to be printed (**REQUIRED**)
<!--endclickprintinclude-->	Indicates the end of the content to be printed (**REQUIRED**)
<!--startclickprintexclude-->	Indicates that the following content should not be included for print
<!--endclickprintexclude-->	Indicates the end of the content that should be included for print
<!--clickprintexcludeimages-->	Indicates that ALL images within the page should be excluded for print
<!—clickprintNextPageURL (URL)-->	Indicates that there are additional pages following the current page (note your account needs to be configured for multiple pagination before using this tag)

Examples:

```
<!--startclickprintinclude-->
<!--clickprintexcludeimages-->
Article text here
<img src="image1.gif">
Article text here
<img src="image2.gif">
Article text here
<img src="image3.gif">
<!--startclickprintexclude-->
Images by Joe Photographer
<!--endclickprintexclude-->
Article text here
<!--endclickprintinclude-->
```

In the case of a piece of content with 3 pages, the first page might include the tag,

```
<!-- clickPrintNextPageURL http://www.mydomain.com/article.html?page=2 -->
```

The second page would include the tag,

```
<!-- clickPrintNextPageURL http://www.mydomain.com/article.html?page=3 -->
```

The third (and final) page would not include this tag.

For detailed instructions on how to mark up your Content page HTML to include/exclude various portions in the “Print This” version of the page, go to **http://community.clickability.com > Knowledge Base > Interactivity Tools > How-To: Configure the Print This Interactive Tool.**

MOST POPULAR results

mpQuery	**$imware.mostPopular2**([int *destid*,] String *toolname,* String *period [,* int *count])*
mpQuery	**$imware.mostPopular2**([int *destid*,] String *toolname,* String *period,* int *count*, Date *date)*

Given a mpQuery object: $myMpQuery:

ArrayList of mpItems	$myMPQuery.results
Date	$myMPQuery.minDate
Date	$myMPQuery.maxDate
Int	$myMPQuery.destID
Object	$myMPQuery.partner

Given a mpItem object: $myItem

String	$myMPItem.url
String	$myMPItem.title
String	$myMPItem.count
Long	$myMPItem.contentID
Int	$myMPItem.contentDestID

Usage:

- **destid** – the ID of the section (and child sections) for which most popular data should be retrieved.
- **toolname** – the type of most populardata to be retrieved. See table below. This is case insensitive.
- **period** – the time period for which to retrieve data. See table below. This is case insensitive.
- **count** – the number of items to return (default is 10)
- **date** – a Date object representing the end date (and time) of the period to be measured. Note these Date-time objects are timezone specific.

Use $imware.mostPopular2 to include MOST POPULAR lists onto pages published by the Clickability Platform. The mostPopular2 tags are used to retrieve an ArrayList of mpItems that contain the titles, urls, counts, and ranks of the result items. Note the logic that outputs the ArrayList should verify that the items are live before linking to them, since the mpItems can include items that are no longer live.

Having an $imware.mostPopular2 tag on your page will ensure the page gets rebuilt hourly in order to get the most current data (aggregated hourly).

Data To Retrieve	toolname
Most Emailed	E
Most Saved	S
Most Printed	P
Most Viewed	V
Most Commented	C
Most Rated	R

Time Period	period
Last 24 hours	D
Today since midnight, customer timezone	d
Last 7 days	W
This week since 12:01am Monday, customer timezone	w
Last 30 days	M
This month since 12:01am on 1st of month, customer timezone	m
Last [complete] Hour	H

Examples:

```
$imware.mostPopular2(5126,"V","D").results
```

- would return an ArrayList of the top 10 most viewed items from destination ID 5126 for the last 24 hours.

```
$imware.mostPopular2(5126,"E","W",25).results
```

- would return an ArrayList of the top 25 most e-mailed items from destination ID 5126 for the last 7 days.

```
#set($endDate = $util.newDate(2006,1,5))
$imware.mostPopular2("V","D",10,$endDate).results
```

- would return an ArrayList of the top 10 most viewed items from the current destination for the 24-hour period ending at 0:00 (in your timezone) on Jan 5, 2006.

Building on the example above, the following code would display a link to each most-viewed item, followed by the # of pageviews it received during that period:

```
#foreach($mpItem in $imware.mostPopular2("V","D",10,$endDate).results)
       <a href="$mpItem.url">$mpItem.title</a> ($mpItem.count)
#end
```

Tracking

Page Tracking

String	$imware.tracker
String	$imware.tracker(long *destid*)
String	**$imware.tracker**(boolean *useClickJsVar*)
String	**$imware.tracker**(long *destid*, boolean *useClickJsVar*)
String	$imware.useStandards()

This tag outputs tracking code needed for the Clickability Platform reports at the page level and higher: e.g., Pageviews, Visits, System reports, Newsletters, etc. It should therefore be included on every page you want tracked, with the one exception of TEXT newsletter templates, since the tag outputs javascript (or in HTML emails, an image tag).

$imware.tracker should ideally go immediately after the <body> tag.

The Tracker sets a JavaScript variable labeled as "js" which can be problematic if external JavaScript uses the same variable name. In that case it is possible to pass 'true' for $imware.tracker to set a "click_js" variable instead of a generic "js" variable.

Note: in some cases it is necessary to use the Tracker with JavaScript that has newer standards compliancy, specifically for the "type" attribute (which is set as the "language" attribute in the regular $imware.tracker JavaScript). To use the "type" attribute instead, call $imware.useStandards() prior to calling $imware.tracker.

URL Tracking

String (URL)	**$imware.trackedUrl**(String *linkURL*, String *title*)
String (URL)	**$imware.trackedUrl**(MediaObject media) * **Note:** this method does not work with Email Newsletters; see below for a workaround
String (URL)	**$imware.trackedUrl**(WebsiteSectionObject *section)*
String (URL)	**$imware.trackedUrl** (ContentObject *content)*
String (URL)	**$imware.trackedTag**(MediaObject or MediaPlacement *media)*
String	$imware.disableLinkTracking()

Usage:

- **linkURL** – the url of the external page you want to track links to
- **title** – the label with which you want the linkURL to appear in link tracking reports
- **media** – the MediaObject for the item you want to track links to
- **section** – the WebsiteSection whose landing page you want to track links to
- **content** – the Content item whose detail page you want to track links to

$imware.trackedUrl can be used in both newsletters and on websites to track activity on links to various WCM objects (destination, content, and media objects) as well as to specific URLs (including external URLs). It outputs a tracking URL that routes the original URL through a special Clickability Platform tracking page. (In newsletter pages, these tracking URLs begin http://s.clickability.com/s. In all other pages they begin http://www.clickability.com/r.) Note that these output URLs will NOT be functional in preview mode.

Examples:

```
<a href="$imware.trackedUrl('http://www.cnn.com','cnn')">News</a>
```

- tracks the activity for a link to www.cnn.com (labeled in link activity report as “cnn”).

```
<a href="$imware.trackedUrl($cms.media('sales.doc', 'documents') ">Download
Here</a>
```

- tracks the activity for a link to the “sales.doc” document.

```
#set($pdf = $cms.content.mediaPlacement1)
<a href="$imware.trackedUrl($cms.media($pdf.id))">Download PDF Here</a>
```

- tracks the activity for a link to a content item’s Media Placement. Note that this method requires a MediaObject (vs. the MediaPlacement returned by the media placement) so $cms.media(int mediaID) is used.

```
#foreach($story in $cms.links("/news",1,1))
      <a href="$imware.trackedUrl($story)">$story.title</a>
#end
```

- tracks the activity for a link to the detail page of the $story content item.

Rendering tracked images for media placements that have Target URLs

The $imware.trackedUrl(MediaObject) method above is for tracking links directly to a media asset. To instead link a Media Placement image to its Target URL, and track that without having to manually construct the <a href> and <img> tags, you can use the $imware.trackedTag method to automatically output the correct linked, tracked image, like this:

```
$imware.trackedTag($cms.content.mediaPlacement1)
```

Viewing Activity Reports for Tracked Links

Tracked link activity can be found in these 3 separate reports, depending on the usage:

- **Media Downloads**
  Found under Site Traffic > Media Downloads, this reports the activity on any tracked direct links to media assets (i.e., links tracked via $imware.trackedUrl(MediaObject)).

- **Newsletter Tracked Links**
  Found under Newsletters > Newsletters, in the “Clickthough URL” column, this reports the activity on all tracked links from newsletters EXCEPT for Media Downloads (as explained above).

- **Website Tracked Links**
  Found under Marketing > Offsite Link Clicks, this reports the activity on all tracked links from the website EXCEPT for Media Downloads (as explained above).

Avoiding conflicts with onClick link tracking
Due to the need for an onClick event for certain types of link tracking, customers have had difficulty on some pages where onClick events were being used at the same time as $imware.tracker calls. To use the Tracker without overriding other onClick actions, use $imware.disableLinkTracking(), which should be called prior to the $imware.tracker call.

Note: doing so will result in no longer tracking data for the following types of events:

- Exit URLs
- Exit Domains
- Offsite Link Clicks

Tracking Links to Media Objects within Email Newsletters

The $imware.trackedURL(MediaObject media) method does not function properly with Email Newsletters unless you use the following workaround:

```
#set($mediaObject = $cms.media("$mailing.mediaPlacementName","mediaType"))
#set($trackedURL =
"http://${cms.envHost}${cms.domain}$imware.trackedURL($mediaObject)")
```

Interactive Tools Tracking

Though specific Interactive tool activity is tracked by the Interactive Tools applications themselves, to track standard, page-level data on the Clickability Platform pages containing $imware.buttons or $imware.customButtonCode, **you must still use** $imware.tracker.

Sending Mail

EmailObject	$util.newEmail

Given an Email object $myEmail:

$myEmail.setUseUnicode(true)

$myEmail.setTo(string email-addresses)

$myEmail.setFrom(string email-addresses)

$myEmail.setReplyTo(string email-addresses)

$myEmail.setCC(string email-addresses)

$myEmail.setSubject(string str-subj)

$myEmail.setBody(string str-body)

$myEmail.setHTMLBody(string str-htmlbody)

$myEmail.addLine(string str-line)

$myEmail.send

$myEmail.sendHTML

$myEmail.sendMultipart

Usage:

- **email-addresses** – A string containing the desired email addresses.
- **str-subj** – The text to include as the subject or body of the email
- **str-body**– The text to include as the body of the email
- **str-htmlbody**– The text to include as the body of an HTML email
- **str-line** – Text to display (in the body) on its own line.

The Mail tags are used to construct and send an email. A form may be used to post to a template using these tags to allow a reader to send a letter to the editor or similar feature.

Create a new mail object

```
#set($myEmail=$util.newEmail)
```

By default, email objects are encoded as ASCII. If you need to use UTF-8 encoding instead, include the following code:

```
$myEmail.setUseUnicode(true)
```

Add the To and From Addresses:

```
$myEmail.setTo("frank@beans.com")
$myEmail.setFrom("PBJ <peanutbutter&jelly.com>")
$myEmail.setCC("a1@steaks.com, mustard@kraft.com")
```

Construct the Mail Message

```
$myEmail.setSubject("A message from a friend")
$myEmail.setBody(req.param("subject"))
```
- get the body from the query string

If you would like to add a line of text followed by a line break:

```
$mymail.addLine("Don't spam people.")
```

A blank link would simply be:

```
$myEmail.addLine("")
```

Send the Email

There are three options to send the email:

```
$myEmail.send
```

will send the email as a plain text email,

```
$myEmail.sendHTML
```

will send the email as an HTML email, and

```
$myEmail.sendMultipart
```

will send the email as multipart, allowing the email client to choose which version of the newsletter to process.

Note: $myEmail.sendMultipart will send the text specified in $myEmail.setHTMLBody as the HTML email portion, and the text specified in $myEmail.setBody as the text email portion. $myEmail.sendHTML will send the text specified in $myEmail.setHTMLBody, unless that method was not used to specify text, in which case it will send the text specified in $myEmail.setBody.

Content Submissions

The content submission module can be used to allow website visitors to submit content into the Clickability Platform. The content will be delivered into a specific Submission Queue with in the system, from where it can either be deleted or converted into a pre-defined type of Content.

Creating a web submission form

The form must be submitted with a POST request to the action “/m”.

```
<form name="submissionForm" method="POST" action="/m" >
```

The following fields are required to indicate which queue to place the submission in and what type of content it represents. Both the queue id and content type id can be located in the Clickability Platform.

```
<input type="hidden" name="qid" value="1234”>
<input type="hidden" name="ctid" value="4321”>
```

After the submission is processed, the submitter will be redirected back to the website. You are required to specify the path to where the user will be sent

```
<input type="hidden" name="path" value="/return/to/here">
```

Specifying content elements values

The individual elements of the content items are passed in the form in the format “element” dot element name. The following inputs add the title, summary and text elements in the content item.

```
<input type="text" name="element.title" /><br/>
<input type="text" name="element.summary" /><br/>
<input type="text" name="element.text" /><br/>
```

IMPORTANT: The title element is required. Without it, the form will fail to produce a Clickability Platform Submission item.

Currently complex field elements are not supported.

Yes/No elements

The Yes/No fields in the Clickability Platform are true/false fields, so you have to post a value of “true” (rather than “on”, which is what checked checkbox fields pass by default). Ex:

```
<input type="checkbox" name="element.showTitle" value="true" />
```

Linked Content elements

You can populate one (and only one) linked content field on the content item you are submitting. To do so, include the following form fields:

Name	Value
linkedcontent.templatetag	Tagname of linked content
Linkedcontent.[templatetag].appendOnly	If false, replaces existing linked content items; if true, adds on to them.
contentID	The content item to which you want to relate other content items [specified in the tocontentids field]. Note: on a detail page, you may omit this and it will by default just use the current content item.
linkedcontent.[templatetag].tocontentids *	The IDs of the content items to which you want to relate the content item you're submitting.
*This accepts valid Content IDs. If multiple IDs, they should be delivered as if they belong to a multi-select box on the page in question. The items must be valid Content Items in your account, and must be of a type that is in the Linked Content definition. If these conditions are not met, an APIException error will be thrown and nothing will be saved.	

Example

Assuming a linked content field tagged "moviesAndReviews", where you want to let the submitter choose which Movie to relate their [being-submitted] Review to:

```
<input type="hidden" name="linkedcontent.templatetag"
value="moviesAndReviews"/>

<input type="hidden" name="linkedcontent.moviesAndReviews.appendonly"
value="false" />

Choose the Movie you want to relate this Review item to:
<select name="linkedcontent.moviesAndReviews.tocontentids" multiple="true">
     <option value="15045">Rush Hour</option>
     <option value="15044">Hero</option>
     <option value="15043">Rumble in the Bronx</option>
</select>
```

Or, if you wanted the [being-submitted] review to automatically relate to a specific Review content item, use a hidden "tocontentids" field, rather than a select, like this:

```
<input type="hidden" name="linkedcontent.moviesAndReviews.tocontentids"
value="15045">
```

Specifying categorization values

Submission forms can also pass Category values as fields named in the format “category” dot Category Set id, whose values are given as Category ids.

Examples:
This code lets the user set a single category value for the “Level” category set (ID 567):

```
Select a level:
<select name="category.567" />
      <option value= "2441">Federal</option>
      <option value= "2442">State</option>
<option value= "2443">Local</option>
</select>
```

This code lets the user set multiple categories for the “Location” category set (ID 471). Note that in order for submissions to be used to set more than one category for a given category set, that category set must already be configured in the Clickability Platform to allow multiple categories:

```
Location (check all that apply):
<input type="checkbox" name="category.471" value="1316" />New York<br/>
<input type="checkbox" name="category.471" value= "1319" />California<br/>
<input type="checkbox" name="category.471" value= "1321" />Florida<br/>
```

Specifying target values

Submission forms can also specify destinations to which the submission (upon conversion to a content item) should automatically be targeted. This is done by passing the destination IDs as values of one or more “destid” parameters:

Examples:
The following hidden fields ensure that the submissions (upon conversion to content) will be automatically pre-targeted to destid’s 723 and 724.

```
<input type="hidden" name="destid" value="723">
<input type="hidden" name="destid" value="724">
```

Specifying Calendar schedule values

Submission forms can also be used to populate a [single] schedule on the content item's Calendar tab (provided the content type has "enable events" checked). Depending on what type of schedule the form allows them to create (weekly, monthly, etc), different combinations of the following fields will be required:

Form Field	Description	Required	Possible Values
event.type	The type of event recurrence. Specify as a string with one of the following values: 'ONCE', 'DAILY', 'WEEKLY', 'MONTHLY_DAY_OF_WEEK', or 'MONTHLY_DAY_OF_MONTH'. Other fields must be set depending on what type of schedule is selected.	Yes	MONTHLY_DAY_OF_MONTH
event.startDate	The first day of the recurrence. Must be in YYYY-MM-DD format. For ONCE events, startDate should equal endDate.	Yes	2007-03-23
event.endDate	The last day of the recurrence. Must be in YYYY-MM-DD format. For ONCE events, startDate should equal endDate.	Yes	2007-12-12
event.startTime	The start time of the reccurrence. Must be in HH:MM:SS with hour = 00-23, minute = 00-59, second == 00. *Required with an event.endTime.	No	23:03:00
event.endTime	The end time of the recurrence. Must be in HH:MM:SS with hour = 00-23, minute = 00-59, second == 00.	No	00:05:00
event.weeklyDays	The days of the week, as a comma separated list (no spaces!). Values must be full day name in ALL Caps. *Required if event.type = WEEKLY	No	SUNDAY,WEDNESDAY
event.monthlyOrdinal	*Required if event,type == MONTHLY_DAY_OF_WEEK and must be specified as one of FIRST, SECOND, THIRD, FOURTH, LAST.	No	FIRST
event.monthlyDayOfWeek	*Required if event,type ==	No	THURSDAY

	MONTHLY_DAY_OF_WEEK and must be the day of the week in all Caps, such as SUNDAY or WEDNESDAY		
event.monthlyDay	*Required if the event.type == MONTHLY_DAY_OF_MONTH and is the numeric day of the month from 1-31.	No	31
event.location	The event location. Can be any text.	No	
event.details	The event details. Can be any text.	No	

Including Media files in a web submission form

To allow media files to be included in a web submission, first make sure the form is submitted via a POST request to the action "/m," with the encryption type set as follows:

```
<form name="submissionForm" method="POST" action="/m"
enctype="multipart/form-data" >
```

Then provide a media file chooser within the form. The name of each media file chooser must indicate which media placement (of the content type being submitted) it should map to, using the format: placement.[mediaPlacementName].

Examples:

```
Top Image: <input type="file" name="placement.topImg" />
Document: <input type="file" name="placement.doc1" />
Audio file: <input type="file" name="placement.audio1" />
```

The submission can also be used to specify attribute values for the media placement using text fields. The name of the field should indicate which attribute of which media placement, using the format:

```
placement.[mediaPlacementName].[mediaPlacementProperty].
```

For example, for the topImg media placement submitted above:

```
Caption: <input type="text" name="placement.topImg.caption" />
AltText: <input type="text" name="placement.topImg.altText" />
Title: <input type="text" name="placement.topImg.title" />
Author: <input type="text" name="placement.topImg.author" />
Target: <input type="text" name="placement.topImg.target" />
TargetURL: <input type="text" name="placement.topImg.targetURL" />
```

Constructing a Submission Notification Template

SubmissionObject	$submission
Given a Submission object $submission:	
	$submission.title
	$submission.email
	$submission.data
	$submission.element(string elementName)
	$submission.domainID
	$submission.queueName
	$submission.queueID
	$submission.contentTypeName
	$submission.contentTypeID
	$submission.url
	$submission.event("startDate")*
	$submission.event("endDate")*
	$submission.event("details")*
	* $submission.event methods will only work if Calendar tab information was collected by the content submission form.

Usage:

- **elementName** – The suffix of the “element.fieldname” form field

If you choose to create an HTML email notification template to be sent when a Submission is received, the $submission tag will give that template access to the submission elements and information, as well as its URL so you can link to it.

There are built-in tags to access the title (from the required “element.title”) and email (should an “element.email” field be among the submission inputs) elements.

```
$submission.title
$submission.email
```

Any other submission elements can be accessed using the syntax:

```
$submission.element("elementName")
```

For example, given a form field

```
<input type="text" name="element.comment" />
```

You can display that value in the notification template using:

```
Comment: $submission.element("comment")
```

Alternately, you can output the entire set of element name-value pairs as a URL-encoded querystring, using:

```
$submission.data
```

The template can also display information about the submission, such as…

which domain it was submitted from:

```
$submission.domain
```

which queue it was submitted to:

```
$submission.queueID
$submission.queueName
```

which content type the submission (if approved) will ultimately map to:

```
$submission.contentTypeID
$submission.contentTypeName
```

The template can also link to the submission item, as follows:

```
<a href="$submission.url">View Submission</a>
```

Feeds

The Clickability Platform Template Language allows for the retrieval and embedding of RSS feeds into your website pages.

RSSChannel	$rss.channel(string url) **Note:** socket timeout is 10 seconds.
Given an RSSChannel object $myFeed:	
String	$myFeed.title
String	$myFeed.description
String	$myFeed.image.title
String	$myFeed.image.link
String	$myFeed.image.height
String	$myFeed.image.width
String	$myFeed.copyright
String	$myFeed.language
Date	$myFeed.lastBuildDate
Date	$myFeed.pubDate
String	$myFeed.site
Set of RSSItems	$myFeed.items
Int	$myFeed.items.size()
Given an RSSItem Object $myItem:	
String	$myItem.title
String	$myItem.link
String	$myItem.description

Usage:

- **url** – The URL address of the RSS Feed to be retrieved.

Retrieve the RSS Feed

To retrieve an RSS feed, simply pass in the web address of the feed to the $rss.channel tag:.

```
#set($myFeed = $rss.channel("http://rss.news.yahoo.com/rss/topstories"))
```

Display Channel Elements

There are several channel elements which can be accessed and displayed. Not every feed supports every element listed. For example,

```
$myFeed.title<br/>
$myFeed.description<br/>
$myFeed.language<br/>
$myFeed.lastBuildDate<br/>
<br/>
$myFeed.image.title<br/>
$myFeed.image.link<br/>
$myFeed.image.height<br/>
$myFeed.image.width<br/>
```

would output:

- Yahoo! News - Top Stories
- Yahoo! News - Top Stories
- en-us
- Mon May 10 14:18:37 PDT 2004

- Yahoo! News
- http://news.yahoo.com/
- 18
- 142

Displaying Feed Items

The individual items within an RSS feed are retrieved as an array list. These items can be iterated over to display the title, links and summaries:

```
#foreach($item in $myFeed.items)
       <p><a href="$item.link">$item.title</a><br/>
       $item.description</p>
#end
```

Generating Feeds

The above methods demonstrate how to retrieve and process feeds for any RSS url. Additionally, the Clickability Platform can generate RSS feeds from website content.

Note: for both types of generated feeds, the elements for each RSS <item> will pull from the following specifically-tagged elements on the content item, so be sure your content is defined using these tagnames:

“title” for <title>
”summary” for <description>

1) Website Section-based “Out-of-the-box” Feeds

As we saw in the “Destinations” chapter, the Clickability Platform offers a “Publish RSS” checkbox option at the website section level that will, if checked, make a simple“index.rss” feed available at that section’s path.

Feed URL	http://domain/path
Feed Format	/index.rss (rss 0.92) /index.rss2 (rss 2.0) /index.atom (Atom) /index.rdf (rdf)
Feed Size	s=(int size)
Feed Content Type	ct=(int content type ID)

Usage:

- Add the desired feed format to the feed url.
- To specify the list size, add the parameter “s=(int size)”. The default is 10.

To specify a single content type to be included in the list, add the parameter “ct=(int content type ID)”. The default is to include all.

Example:

- http://www.mydomain.com/news/index.rss2?s=25&ct=4025

2) Custom-configured “Outbound” Feeds

Custom Outbound feeds offer additional configuration possibilities including date ranges, categories and various website sections.

Set of OutboundFeed objects	$cms.outboundFeeds
Set of OutboundFeed objects	$cms.getOutboundFeeds(CategorizationCriteria feedCategorization)
OutboundFeed object	$cms.getOutboundFeed(int feedID)
OutboundFeed object	$cms.getOutboundFeed(string urlname)

Given an OutboundFeed object $myFeed:

Object	$myFeed.attribute-name

Usage:

- **feedCategorization** – The CategorizationCriteria you want to match feeds against
- **feedID** – the ID of the Outbound Feed you want to retrieve
- **urlName** – the Resource URL Name of the Outbound Feed you want to retrieve
- **attribute-name --** the name of the feed attribute to be output

Attributes:

- **ID (or feedID)** – the ID of the OutboundFeed
- **customerID**
- **domain, domainID**
- **destID** – a destID unique to the OutboundFeed (not a section id)
- **feedType** – an int representing the type of feed
- **feedSize** – the max # of items the feed will show
- **name** (or title)
- **urlName** -- the Resource URL Name of the OutboundFeed
- **url** (or feedURL)
- **link** – the webpage url associated with your feed
- **createDate**
- **lastUpdated**, updateDate
- **lastModified**, modifiedDate
- **syndicationFeed**

Given a SyndicationFeed object $syndicationFeed

Object	$syndicationFeed.attribute-name

Attributes:

- **entries** – an individual entry within a syndication feed

Given a list of feed entries (obtained with $myFeed.syndicationFeed.entries) the following attributes are available:

- **link** (string)
- **title** (string)
- **author** (string)
- **publishedDate** (date)
- **description** (object) which has two attributes: value (string), type (string)

Example Syndication Feed

```
#foreach($obFeed in $cms.outboundFeeds)
    <a href="$obFeed.url">$obFeed.name</a><br>
    #set($synFeed = $obFeed.syndicationFeed)
    Here is the syndicationFeed:<br>
    #foreach($syn in $synFeed.entries)
        <a href="$syn.link">$syn.title</a><br>
        - author: $syn.author<br>
        - description: $syn.description.value, $syn.description.type<br>
        - published date: $syn.publishedDate<br>
    #end
#end
```

Other XML Documents

The Clickability Platform Template Language allows for the retrieval and parsing of xml documents.

Document	$feed.getXMLDocument(string url)
Document	$feed.getXMLDocument(string url, int ttlInMinutes)

Given the returned Document object, you can use any of the standard JAVA Document Interface methods to traverse the XML document as needed - e.g. getDocumentElement(), getElementsByTagName(string tagName), etc. A full listing can be accessed here: http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Document.html

Note: This will timeout after 5 seconds if there is no response from the specified URL.

Usage:

- **url** – The URL address of the RSS Feed to be retrieved.
- **ttlInMinutes** – The time the retrieved document object will live in cache

Site Subscriptions

Refer to the Clickability Platform User Community at http://community.clickability.com > Developer Forum > Technical Docs & Webinars for the “Guide: Website Registration and Distribution Subscription Setup”.

Subscribed User Information

User Object	$cms.subscriptionUser
Given a User object $user:	
Boolean	$user.isNew() (checks to see if user has been assigned an ID. Will return false once ID has been assigned.)
	$user.attribute-name (attribute names defined below)
Attributes:	
Int	domainName, email, userName, firstName, lastName, birthDateString, title, company, address1, address2, city, state, postalCode, country, phone, mobile, fax, gender, custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9
String	domainName, email, userName, firstName, lastName, birthDateString, title, company, address1, address2, city, state, postalCode, country, phone, mobile, fax, gender, custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9
Date	birthDate

General Subscription Process Tags

General

String	$subscription.requestedPageURL
Boolean	$subscription.freeRegistration
Content Object	$cms.content
Website Section Object	$cms.websiteSection

Errors

Boolean	$subscription.error
String	$subscription.errorMessage
Int	$subscription.errorCode
String	$subscription.errorDetails

Retrieve Password Email

String	$username
String	$email
String	$password

Registration Form Helpers

ArrayList	$subscription.months $month.name $month.ID
ArrayList	$subscription.years $year.name $year.ID
ArrayList	$subscription.states $state.name $state.ID
ArrayList	$subscription.countries $country.name $country.ID

Secure Media URLs

String	$util.getURL(MediaObj)

Usage:

- **MediaObj** – the object returned by retrieving a Media Object from a media repository with $cms.media()

When images are rendered via $util.tag() on the secure portion of a subscription-based Clickability Platform site, they will be properly output using a secure https:// url.

However, if you try to access the url property of a MediaPlacement or MediaObject directly (e.g., via $cms.media($content.image.id, "images")), it will NOT return a secure url.

To directly get a secure URL for a Clickability Platform media asset, use the $util.getURL method. Ex:

```
$util.getURL($cms.media($content.image.ID),"images")
```

Registration via Social Media SSO

Registered User Object	$cms.registeredUser
Given a Registered User object $user:	
Public String	$user.externalProfileId Method to retrieve the external profile id associated with user profile on an external identity system like Facebook.
Public String	$user.externalLink Method to retrieve a profile url that exposes profile data for the user. For Facebook this url returns a JSON object with data about the user.
Public Boolean	$user.requiresConfirmation An override flag that allows implementers to choose if an email confirmation should be sent to the user, if the customer domain/account is configured to have email confirmation.
Public Boolean	$user.externalSSOUser A utility method that allows a single call for implementers to determine if the user they are working with is an user that has used external SSO system for registration/authentication.
Public String	$user.externalData Method to extract the additional data that is stored about the user with the registered user record (preferably in JSON format) as a string.
Public IJSONObject	$user.externalDataJson If the external user data has been stored as a JSON string, this method extracts the additional data as an IJSON object. Implementers can use the $jsonParser object to manipulate the IJSONObject.

Site Search

Content Site Search tags

SiteSearch Object	$cms.newSearch
Given a SiteSearch object $mySearch:	
	$mySearch.addKeywords(string search-keyword(s))
	$mySearch.addBrowserContentID(int contentID)
	$mySearch.addCategorization(CategorizationCriteriaObject cat-obj)
	$mySearch.addContentType(string contenttype-name)
	$mySearch.excludeContentType(string contenttype-name)
	$mySearch.addWSS(string website-section-name|website-path [, Boolean include-children?])
	$mySearch.addWSS(int website-section-ID [, Boolean include-children?])
	$mySearch.addWSS(WebsiteSectionObject website-section-obj [, Boolean include-children?])
	$mySearch.addDateRange(string datefield, date date1, date date2)
	$mySearch.addBeforeDate(string datefield, date date1)
	$mySearch.addAfterDate(string datefield, date date1)
	$mySearch.setDateRange(string datefield, date date1, date date2)
	$mySearch.setBeforeDate(string datefield, date date1)
	$mySearch.setAfterDate(string datefield, date date1)
	$mySearch.setSearchFields(string field-names)
	$mySearch.boost(string field-name, int boostfactor)
	$mySearch.requireAllTerms()
	$mySearch.setSearchSynonyms
	$mySearch.setState(string state-name)
	$mySearch.sortByScore()
	$mySearch.sortByDate()
	$mySearch.setSortBy(string sortString [,string secondarySortString])
	$mySearch.page(integer n)
	$mySearch.pageSize(integer m)
	$mySearch.setEnvironment(int environmentCode)
	$mySearch.setTrackForReports(boolean trackForReports?)
	$mySearch.setStemming(boolean useStemming) **Note:** This setStemming tag works only on the following fields: title, summary, and content body

$mySearch.setEventDateRange(DateString fromDate, DateString toDate)

$mySearch.setEventDateRange(int fromYear, int fromMonth, int fromDay, int toYear, int toMonth, int toDay)

$mySearch.setAllowStopWords(boolean allow?)

$mySearch.setDisableCoord(boolean disable?)

$mySearch.setNormalizeScores(boolean normalize?)

$mySearch.setExpandEvents(boolean expand events?)

$mySearch.setExpandEvents(boolean expand events?)

$mySearch.setPaginateExpandedEvents(boolean paginate?)*

$mySearch.setExcludeCHContent(boolean excludeCHcontent?)**

$mySearch.setCHDefinitionID(int hierarchyID)**

$mySearch.setCHDefinitionNodeID(int heierarchyNodeID)**

$mySearch.setCHInstanceID(int instanceID)**

Usage:

- **search-keyword(s)** – the string to be searched for
- **field-names** – all, title/name, author, metadata, slug, content/body, summary, category/categories, eventlocation, eventdetails. Default is "all".
- **boostfactor** – a decimal number (ie. "3", "1.2")
- **datefield** – edit/editdate/modified/modifieddate, create/createdate, live/livedate/contentlivedate, archive/archivedate/contentarchivedate, dead/deaddate/contentdeeaddate, startdate (will work against any date or date-time field tagged "startDate, startdate"), enddate (will work against any date or date-time field tagged "endDate, enddate")
- **website-section-name** – The name of the website section from whose placement-list the content should be retrieved.
- **website-path** – The URL path to the website section from whose placement-list the content should be retrieved
- **website-section-ID** – The ID of the website section from whose placement-list the content should be retrieved
- **website-section-obj** – The website section (obtained by a $cms.websiteSection() tag) from whose placement-list the content should be retrieved
- **include-children?** – whether or not to include the children, grandchildren, etc. of the specified website section. Acceptable values are true and false.
- **contenttype-name -** The name of a content type defined in the Clickability Platform
- **state-name** – either archive or live
- **environmentCode** – 0 for dev, 1 for stage, 2 for production
- **cat-obj**– a catergorization object as specified in the "Content tags" chapter.
- **sortString** – the SearchResultObject field by which to sort results
- **use-stemming?** – set to false if you don't want to use word stemming. Default is true.
- **trackForReports?** – set to false if you don't want data from this search to be included in reports. Default is true.
- **setAllowStopWords** – set to true if you want stop words like 'a', 'the', 'them', 'it' to be included in search results data. Default is false. A list of all stop words is available at http://community.clickability.com.
- **setDisableCoord** – Set this to false and search result scores will be lower. Default is true.

- **setNormalizeScores** – Set this to true to use legacy scoring algorithms. Default is false.
- **excludeCHContent**** – Set this to true to exclude content items tied to any hierarchy. Default is false.
- **hierarchyID**** - This will limit the search results to a particular content hierarchy definition.
- **heierarchyNodeID**** - This will limit the search results to a particular node of a hierarchy definition.
- **instanceID**** - This will limit the search results to a particular instance of a hierarchy.

***Note:** Can be used ONLY if setExpandEvents(true)

****Note:** These tags are only applicable to content enabled with the Content Hierarchy feature. Content Hierarchy provides a way to associate groups of content together.

ArrayList of SiteSearchResults	$mySearch.execute
Int	$mySearch.estimatedPages
Given a SiteSearchResults object $searchResult:	
Int	$searchResult.score
Int	$searchResult.id
Int	$searchResult.typeID
String	$searchResult.typeName
String	$searchResult.title
String	$searchResult.author
Date	$searchResult.createDate
Date	$searchResult.modifiedDate
Date	$searchResult.contentLiveDate
Date	$searchResult.startDate
Date	$searchResult.endDate
String	$searchResult.summary
String	$searchResult.slug
String	$searchResult.url
String	$searchResult.urlName
String	$searchResult.path
Int	$searchResult.destID
String	$searchResult.stars
String	$searchResult.stars(string color)
ArrayList of ConetntTarget Objects	$searchResult.contentTargets
ArrayList of CalendarSchedule Objects	$searchResult.calendarSchedules
CalendarSchedule Object	$searchResult.schedule*
Occurrence Object	$searchResult.event*

Usage:

- **color**– one of eight colors: blue, purple, green, red, yellow, orange, tan, gray.

*__Note:__ Can be used **ONLY** if setExpandEvents(true)

Context Site Search tags

Context Search Object $cms.newSearch

Given a Context Search $searchContext:

$searchContext.addKeywords(String keywords, String searchFields)

$searchContext.addKeywords(String keywords, String searchFields, boolean required)

$searchContext.addKeywords(String keywords, List searchFields)

$searchContext.addKeywords(String keywords, List searchFields, boolean required)

Usage:

- **keywords** – The text keywords to search for within the specified fields. Keywords should be separated by spaces, as in the existing addKeywords method.
- **searchFields (String) -** A comma separated list of fields that the specified keywords should be applied to. Valid field names are the same as for the exising setSearchFields(...) methods.
- **searchFields (List of Strings) -** An array list of search fields specified as strings. Provides an alternative to passing in a comma separated list.
- **required -** Whether one of the terms must match to one of the fields specified within the terms and fields specified in each specific call. If false is specified (the default) then this method generally acts to enhance the scores of results to which these keywords are matched.

Notes:

- In each case the parameters are defined above. If a required parameter is not specified, the default will be FALSE.
- The existing addKeywords(String keywords) has been left unchanged and should continue to work as it always has, even in conjunction with these new fields. The existing addKeywords fields acts upon the fields set via the $searchContext.setSearchFields(...) family of methods.

Media Site Search tags

MediaSearch Object	$cms.newMediaSearch
Given a MediaSearch object $mySearch:	
	$mySearch.addKeywords(string search-keyword(s))
	$mySearch.setSearchSynonyms
	$mySearch.addCategorization(CategorizationCriteriaObject cat-obj)
	$mySearch.addMediaType(string mediatype-name)
	$mySearch.excludeMediaType(string mediatype-name)
	$mySearch.setDateRange(string datefield, date date1, date date2)
	$mySearch.setBeforeDate(string datefield, date date1)
	$mySearch.setAfterDate(string datefield, date date1)
	$mySearch.setSearchFields(string field-names)
	$mySearch.boost(string field-name, int boostfactor)
	$mySearch.requireAllTerms()
	$mySearch.sortByScore()
	$mySearch.sortByDate()
	$mySearch.page(integer n)
	$mySearch.pageSize(integer m)

Usage:

- **search-keyword(s)** – the string to be searched for
- **field-names** – all, name, title, author, metadata, creatorid, caption, filename, categories, category, content/doccontent/documentcontent
- **boostfactor** – a decimal number (ie. “3”, “1.2”)
- **datefield** – edit/editdate/modified/modifieddate, create/createdate, live/livedate/contentlivedate, archive/archivedate/contentarchivedate, dead/deaddate/contentdeeaddate
- **mediatype-name -** image/images, video/videos, doc/docs/document/documents, binary, audio
- **cat-obj** – a catergorization object as specified in the “Content tags” chapter.

ArrayList of MediaSearchResults	$mySearch.execute
Int	$mySearch.numPages
Given a MediaSearchResults object $searchResult:	
Int	$searchResult.score
Int	$searchResult.id

Int	$searchResult.typeID
String	$searchResult.typeName
String	$searchResult.title
String	$searchResult.author
String	$searchResult.credit
Date	$searchResult.createDate
Date	$searchResult.modifiedDate
String	$searchResult.filename
String	$searchResult.extension
String	$searchResult.altText
String	$searchResult.metadata
String	$searchResult.description
String	$searchResult.caption
String	$searchResult.copyright
String	$searchResult.size
String	$searchResult.height
String	$searchResult.width
String	$searchResult.url
String	$searchResult.stars
String	$searchResult.stars(string color)

Usage:

- **color** – one of eight colors: blue, purple, green, red, yellow, orange, tan, gray.

BEST PRACTICE

Locate your search result pages at “/search” or “/searchresults”, since the robots file for your site will by default already be configured to keep web-crawlers from accessing those paths.

If you need the search result page to be located elsewhere, you can still modify your robots file to keep crawlers out of that location.

Performing a Site Search Query

The tools available to perform a Site Search function for your readers are very flexible. There are several steps to creating a query and returning the results:

Create the Search Object

For content:

```
#set($mySearch=$cms.newSearch)
```

For media:

```
#set($mySearch=$cms.newMediaSearch)
```

Query String

Search keywords are added to the search object a tag such as the following:

```
$mySearch.addKeywords("pies")
```

or,

```
$mySearch.addKeywords("blueberry pies")
```

Multiple word strings may be passed to the search engine and when the search engine performs the query, it will return a record whenever it encounters one or more of the keywords. Note multiple word strings should be passed with only ONE $mySearch.addKeywords call; if multiple calls are used, the words in the last call will be the only words searched for.

Boolean operators are not currently supported.

Optionally, synonyms can be added to the keywords, such that if you enter “big”, it will also search for “large.” Include synonyms via the following tag:

```
$mySearch.setSearchSynonyms
```

Set Search Fields

The site search can be performed on any combination of available* search fields. For content items, these fields include the author, title, summary (any content item field with a tagname of *summary*), metadata (any content item field with tagname of *metadata*), slug** (any content item field with tagname of *slug*), content (all non-multi-valued fields except for the title, author, summary, slug, metadata), and categories.

Note**: The following types of content fields are not indexed and ***CANNOT be searched: select lists, complex fields, item browsers, Yes/No fields, and non-system date & date/time fields.

****Note about the *slug* field**: If a content item uses a one-line text field with a template tag of *slug*, this field will be indexed for searching. The search done on this field will be a prefix search, meaning that any slug which begins with the search term will be returned. For example, if "fr" was entered, items with "Fred", "friend", "fr123", "France" in the slug field would all be returned in the search results.

For media, the available search fields include the author, title, name, filename, metadata field (any content item field with a tagname of *metadata*), caption, creatorID, content (all non-multi-valued fields except for the title, summary and metadata), and categories.

The setSearchFields tag uses an array of strings to identify which fields the search engine should utilize in its search. The possible field designators are:

Field Designator - Content
all (includes title, metadata, slug, summary, categories, content, eventlocation* and eventdetails*)
title/name
metadata
slug
summary
content/body
category/categories
author
eventlocation*
eventdetails*

Field Designator - Media
all (includes title, metadata, filename, categories, content, caption)
title/name
metadata
filename
content / doccontent / documentcontent
category/categories
author
creatorid
caption

* For content types with Calendar Schedules

The default behavior is to use **All**. The designators are not case sensitive, but any error in definition (ie a spelling mistake) will revert to the default value.

Examples:

```
$mySearch.setSearchFields("body,title")
$mySearch.setSearchFields(["content","SummaRy"])
$mySearch.setSearchFields("title")
$mySearch.setSearchFields("doccontent,filename")
$mySearch.setSearchFields("all,author")
```

Boost

You may “boost” the importance of certain search-designator fields (author, title, metadata, summary) by assigning a boost factor. To boost the Metadata field’s relevance to twice the other fields:

```
$mySearch.boost("metadata", 2)
```

Search Item Browsers for an ID

You can configure the search to look for items that refer (via any of their item browsers) to a particular content id:

```
$mySearch.addBrowserContentID(152803)
```

would return any content items that referenced content item #152803 in any of their item browsers – i.e., you cannot specify any one specific item browser to search in.

Set Require All Terms

The default behavior for a multiple term site search query is to search for items which contain at least 1 of the terms. For example, a search for “lazy dog” would return any item containing “lazy” or “dog”. The requireAllTerms tag is used to require that all terms to be present in each result item, both “lazy” and “dog”.

```
$mySearch.requireAllTerms()
```

The default behaviour is to not require all the terms.

Content Type Filter

You may limit the search to one or more content types using the following tag. The default behavior if no content type(s) are specified is to look for all content types.

```
$mySearch.addContentType("Food Descriptions")
$mySearch.addContentType("Food Reviews")
```

Similarly, you may limit the search to exclude one or more content types (which you should do for Content Types that do not have a template associated with them):

```
$mySearch.excludeContentType("Movie Reviews")
```

These tags may be called multiple times.

Media Type Filter

You may limit the search to one or more media types using the following tag. The default behavior if no media type(s) are specified is to look for all media types.

```
$mySearch.addMediaType("Documents")
```

Or, you may limit the search to exclude one or more media types:

```
$mySearch.excludeMediaType("Video")
```

These tags may be called multiple times.

Website Section Filter

For content searches, you may limit the search to content targeted to one or more website sections. The default behavior is to find content in all website sections. A true / false parameter is used to specify if descendant website sections should be included in the query.

```
$mySearch.addWSS("/foods", true)
$mySearch.addWSS("Recipe Reviews")
```

This tag may be called multiple times.

Categorization Filter

You may limit the search to content of specific categories by including a categorization criteria object in the search. See the “Content tags” chapter for more details on creating categorization criteria objects.

```
#set($catobj=$cms.getCategorization(["Foods"],["Fruit"], true))
$mySearch.addCategorization($catobj)
```

Multiple Categorization Filter

Multiple categorization objects can be included in a search to create AND-OR types of searches. In this configuration, the match-all Boolean determines whether the categorization is added as an AND or an OR.

```
$cms.getCategorization(["setOne"],["A"], match-all?)
```

If match-all is set to true, it is interpreted as AND. If match-all is set to false, it is interpreted as OR. The match-all is ignored on the first category addition.

Example:
The following categories added to search:

```
$mySearch.addCategorization($cms.getCategorization(["SetOne"],["A"], false))
- match-all is ignored on first term

$mySearch.addCategorization($cms.getCategorization(["SetTwo"],["X"], false))
```

Will result in search results where SetOne=A OR SetTwo=X.

The following categories added to search:

```
$mySearch.addCategorization($cms.getCategorization(["SetOne"],["A"], false))
- match-all is ignored on first term

$mySearch.addCategorization($cms.getCategorization(["SetTwo"],["X"], true))
```

Will result in search results where SetOne=A AND SetTwo=X.

Adding another category object to the search will group items:

```
$mySearch.addCategorization($cms.getCategorization(["SetOne"],["A"], false))
- match-all is ignored on first term

$mySearch.addCategorization($cms.getCategorization(["SetTwo"],["X"], true))

$mySearch.addCategorization($cms.getCategorization(["SetTwo"],["Y"], false))
```

And result in search results where (SetOne=A AND SetTwo=X) OR SetTwo=Y.

Creating a group of categories within one categorization object makes it possible to find search results that match some of one category AND some of another. For example, the following logic could be used to find a restaurant, bar OR lounge in San Francisco OR Berkeley.

```
$mySearch.addCategorization($cms.getCategorization(["Entertainment","Enterta
inment","Entertainment"],(["Bars","Restaurants","Lounges"], false))

$mySearch.addCategorization($cms.getCategorization(["Location","Location"],[
"San Francisco","Berkeley"], true))
```

In the example above, you can see that OR logic is always used to combine the multi category objects for search. To achieve AND logic, revert back to the single category categorization object logic. For example, if you wanted to find all Bars that have Happy Hours on Tuesdays in San Francisco OR Berkeley, you would use the following logic:

```
$mySearch.addCategorization($cms.getCategorization(["Entertainment"],(["Bars
"]))
$mySearch.addCategorization($cms.getCategorization(["Entertainment"],(["Happ
y Hours"], true))
$mySearch.addCategorization($cms.getCategorization(["DaysOpen"],(["Tuesday"]
, true))
$mySearch.addCategorization($cms.getCategorization(["Location"],["San
Francisco","Berkeley"], true))
```

Date Filters

You may limit the search to content created or modified on or between specific dates, by indicating either a before/after date, or a date range.

```
#set($date1)=$util.parseDate("01/01/05","MM/dd/yy")
#set($date2)=$util.parseDate("12/31/05","MM/dd/yy")
$mySearch.addDateRange("modifieddate",$date1,$date2)
$mySearch.addBeforeDate("editdate",$date2)
$mySearch.addAfterDate("create",$date1)
```

Note: Multiple 'add' methods can be applied, but only one 'set' method can be applied. Using a 'set' method will overwrite *any* previous date method calls (including 'add' methods). If a search is configured with addDateRange *and* setBeforeDate *and* setAfterDate, the last 'set' method would be all that applies.

Calendar Schedule Filters

You may limit the search to content scheduled (via its Calendar tab) for a particular date range. This will return all content items whose schedule(s) contain one or more occurrences that fall within the specified date range (where the range includes the from/to bounding dates). **Example:**

```
$mySearch.setEventDateRange("2008-01-01","2008-1-31")
```

would return any content items whose Calendar tab included schedules with occurrences on any day in January.

Because a single content item may have more than one scheduled occurrence within a given date range, the search has an option to expand the search results to return one copy of the content item for each occurrence it has within the searched-for date range. In this occurrence-expanded search mode, the single day-date of each occurrence can be accessed via:

```
$result.event.date
```

("event" should be considered synonymous with each individual "occurrence")

For example: Item A is scheduled to occur on Tuesdays and Thursdays while Item B is scheduled to occur on Wednesdays. If you search for all events from Monday to Friday, normally the results would be:

- Item A
- Item B

However if you set the expansion flag $mySearch.setExpandEvents(true), the results would be:

- Item A (where the $result.event.date = tues)
- Item A (where the $result.event.date = thur)
- Item B (where the $result.event.date = wed)

Note: If the results will be paginated, be sure to also call this tag to count each occurrence of an event as an item on the page. Without this tag, each event will count as one item even if it has multiple occurrances.

```
$mySearch.setPaginateExpandedEvents(true)
```

Once expanded, the results can be sorted on the "event.date" field and you can easily construct a calendar on the results!

```
#set($results=$util.sort($mySearch.execute, "event.date"))
```

Content Status

For content searches, you may limit the search to either content that is Archived or Live. The default behavior is to find content in either status.

```
$mySearch.setState("Live")
```

or,

```
$mySearch.setState("Archive")
```

Environment

By default, the search will look only for content items targeted to the environment you're currently on (dev, stage, or production). But if you have a need to explicitly specify in which environment you want it to look, you can do so using:

```
$mySearch.setEnvironment(int environmentCode)
```

where the environmentCodes are 0 for dev, 1 for stage, and 2 for production.

Results Ordering

You may specify if the results are sorted by the document score (highest first), by the lastModified date of the content items (most recent first), or by any field or expression derived from the SiteSearchResult object itself. The default behavior is to sort by score.

```
$mySearch.sortByScore()
```

or,

```
$mySearch.sortByDate()
```

or,

```
$mySearch.setSortBy(sortString)
```

where sortString is a field of the SiteSearchResult object, or some other expression derived therefrom (See the "ArrayLists" chapter for more on sortStrings). Sorts are in ascending "+" order by default, but can be made descending by prepending a "-" in the string.

Examples:

```
$mySearch.setSortBy("title")
```

will sort in ascending order by the results' title field.

```
$mySearch.setSortBy("-contentLiveDate")
```

will sort in descending order by contentLiveDate. Note that for content that's targeted to *more than one* section, the contentLiveDate will be from the section it went live in FIRST.

Note: In the new version of search launched in October 2009, only the following fields are recognized as sortBy fields:

```
startDate
endDate
contentLiveDate
liveDate
createDate
modifiedDate
lastModified
editDate
updatedDate
id
title
```

There is only one recognized secondary sort option:

```
title, -startDate
```

Any other sortBy designations made will revert to our legacy search code and behave as if the $mySearch.normalizeScores(boolean) were set to 'true'.

Note: Any of the above sort tags must be placed before the search is executed.

Keeping Search out of Reports Data

At times your template code may need to run an internal search (i.e, not one driven by site users clicking a Search button) and in those cases you can keep data about that search out of the Clickability Platform search reports by using this tag:

```
$mySearch.setTrackForReports(false)
```

Results Page Size

Search results are returned by pages. You may specify the size of these results pages.

```
$mySearch.pageSize(25)
```

The default page size is 10.

Results Page

Search results are returned by pages. You may specify which page of the results you would like to retrieve.

```
$mySearch.page(4)
```

The default page is 1.

Execute the Query

Executing the query returns an array list of Site Search Results objects, which contain all the basic elements of the content items, but for reasons of efficiency don't actually load them.

```
$mySearch.execute
```

Displaying Site Search Results

Site Search Result Objects

Site Search Result Objects are efficient representations of the content items returned by a site search query. They contain all the basic elements of a content item, without having to load the whole content item itself. These elements are accessed as follows:

```
$searchResult.score
```

- A score of 0-100 based on how well the item matched the search query.

```
$searchResult.id
```

- The contentID of the result item.

```
$searchResult.typeID
```

- The content type ID of the result item.

```
$searchResult.typeName
```

- The content type name of the result item.

```
$searchResult.title
```

- The title of the result item.

```
$searchResult.author
```

- The author of the result item.

```
$searchResult.createDate
```

- The date on which the result item was created.

```
$searchResult.modifiedDate
```

- The last date the result item was modified.

```
$searchResult.url
```

- The URL of the result item.

Note: If the searchResult is targeted to more than one website section, its url will be constructed based on the first section in which the search found a target –there is no way to know which target that will be.

```
$searchResult.stars
```

- Outputs an image tag for an image which graphically represents the result items score. The images are sets of stars. One of eight colors can be specified: blue (default), purple, green, red, orange, yellow, tan, gray, using `$searchResult.stars("red")`

```
$searchResults.estimatedPages
```

- This tag returns the estimated number of pages of results based on the result page size (either when specified or when default page size is used). This is only an estimate because iterating through all the results to make this number accurate would significantly degrade performance.

Content results only

```
$searchResult.summary
```

- The summary of the search result content item. This is populated only if the content type contains an element with “summary” as the tag.

Media results only

```
$searchResult.filename
```

- The filename of the media result item

```
$searchResult.altText
```

- The Alt Text of the media result item

```
$searchResult.metadata
```

- The metadata of the media result item

```
$searchResult.caption
```

- The caption of the media result item

```
$searchResult.size
```

- The file size of the media result item

```
$searchResult.height
```

- The height of the media result item

```
$searchResult.width
```

- The width of the media result item

Site Search Examples:

Search Example #1

The following template code,

```
#set($search=$cms.newSearch)

$search.addKeywords("pies")

#set($catobj=$cms.getCategorization(["Foods"],["Fruit"]))
$search.addCategorization ($catobj)

$search.addContentType("Food Descriptions")
$search.addContentType("Food Reviews")

$search.addWSS("/foods", true)
$search.addWSS("Recipe Reviews")
$search.addState("Archive")

$search.sortByDate()

#foreach($result in $search.execute())
      $results.title<br/>
#end
```

would return the titles of all content items that:

are of "Food Description" or "Food Reviews" contentType

and

are targeted to "Recipe Reviews" or are targeted to "/foods" or child sections of "/foods"

and

are in archive status

and

are categorized in the "Foods" category set as "Fruits"

and

contain the keywords "pies".

Search Example #2

Of course, it is more likely that your search parameters would be passed in the format of HTML Form parameters. The following code illustrates the code for an example of an HTML Form posting to a results template:

Search Form:

```
<form action="/results" method="get">
	Search for:
	<input type="text" name="searchstring"><br/>
	In section:
	<select name="section">
		<option value="/">All</option>
		<option value="/usnews">US News</option>
		<option value="/worldnews">World News</option>
	</select>
<input type="submit">
</form>
```

Results Template:

```
#set($search = $cms.newSearch)
$search.addKeywords($req.param("seachstring"))
$search.addWSS("$req.param('section')", true)
#set($results=$search.execute())
#if($results.size() > 0)
	#foreach($item in $results)
		<a href="$item.url">$item.title</a><br/>
	#end
#else
	Sorry, there were no results.
#end
```

'More Like This' Site Search

The 'More Like This' search function provides search results that are similar to a particular content item. It includes similar search properties as regular Site Search in that it uses default Stop Words and porter stemming. In order to execute a 'More Like This' search, you need to:

1. Have Clickability Platform Client Services activate this module for your account
2. Create a 'More Like This' search object and set it to a Velocity variable
3. Set search parameters by calling the appropriate methods (see table below)
4. Execute the search to obtain a list of SiteSearchResults objects (note they are the same type of object with the same template tags as returned by regular Site Search)

MoreLikeThisSearch Object $cms.newMoreLikeThisSearch

Given a MoreLikeThisSearch object $mltSearch:

$mltSearch.minScore(int min-score)

$mltSearch.maxResults(int max-results)

$mltSearch.addTemplateTagsToAnalyze(string template-tag)

$mltSearch.maxQueryTerms(int max-terms)*

$mltSearch.minTermFreq(int term-frequency)*

$mltSearch.minDocFreq(int doc-frequency)

$mltSearch.minWordLength(int world-length)*

$mltSearch.maxWordLength(int world-length)

$mltSearch.maxNumTokensParsed(int tokens)

$mltSearch.minNumCharsToBeAnalyzed(int characters)*

$mltSearch.addContentType(string contenttype-name)*

$mltSearch.excludeContentType(string contenttype-name)*

$mltSearch.addWSS(string website-section-name|website-path[, Boolean include-children?])

$mltSearch.addWSS(int website-section-ID[, Boolean include-children?])

$mltSearch.addWSS(WebsiteSectionObject website-section-object[, Boolean include-children?])

$mltSearch.addDateRange(string datefield, date date1, date date2)

$mltSearch.addBeforeDate(string datefield, date date1)

$mltSearch.addAfterDate(string datefield, date date1)

$mltSearch.setDateRange(string datefield, date date1, date date2)

$mltSearch.setBeforeDate(string datefield, date date1)

$mltSearch.setAfterDate(string datefield, date date1)

$mltSearch.addCategorization(CategorizationCriteriaObject cat-obj)

$mltSearch.setState(string state-name)

$mltSearch.setEnvironment(int environment-code)

$mltSearch.addInterest(string interest-tag)**

$mltSearch.setRequireAllInterests(Boolean require-all?)**

Usage:

- **min-score** – The minimum score that must be met between 1-100 for a search result to be included in the list of search results (default is 60).
- **max-results** – The maximum number of results that will returned by the search; note this cannot exceed the system-wide maximum (default is 10).

- **template-tag** – Configure which fields on the selected “More Like This” content item will be analyzed to derive the More Like This query term from. Note there is no validation to check if the tag names exist in the content type of the supplied content item. If no values are set, the default will include all strings except for author.
- **max-terms*** – Sets the maximum number of terms that will be included in the “More Like This” generated query (default is 10).
- **term-frequency*** – The number of terms with a frequency below this value within the source content item will be ignored (default is 3).
- **doc-frequency -** The number of terms that appear with a frequency below this value within other content items will be ignored (default is 5).
- **word-length*** – Specifies the number of characters that should determine whether a word is included or excluded in the search. Using 0 has no effect. Default minimum is 4 and there is no default maximum.
- **tokens** – Used to control how many terms are parsed when generating term frequency vectors on the fly, as opposed to precalculated (default is 5000).
- **characters*** – the number of characters that must be found in the source content item (default is 800).
- **contenttype-name*** – The name of a content type defined in the Clickability Platform.
- **website-section-name** – The name of the website section from whose placement-list the content should be retrieved.
- **website-path** – The URL path to the website section from whose placement-list the content should be retrieved.
- **website-section-ID** – The ID of the website section from whose placement-list the content should be retrieved.
- **website-section-obj** – The website section (obtained by a $cms.websiteSection() tag) from whose placement-list the content should be retrieved.
- **include-children?** – whether or not to include the children, grandchildren, etc. of the specified website section. Acceptable values are true and false.
- **datefield** – edit/editdate/modified/modifieddate, create/createdate, live/livedate/contentlivedate, archive/archivedate/contentarchivedate, dead/deaddate/contentdeeaddate, startdate (will work against any date or date-time field tagged “startDate, startdate”), enddate (will work against any date or date-time field tagged “endDate, enddate”)
- **cat-obj**– a catergorization object as specified in the “Content tags” chapter
- **state-name** – either archive or live
- **environmentCode** – 0 for dev, 1 for stage, 2 for production
- **interest-tag ** ** – Returns content items that have this Interest assigned to them.
- **require-all ** ** – Used in conjunction with the addInterest tag to require content items returned to be assigned to the Interest values.

* **Note:** Refer to best practices listed below.

** **WMA-Specific tags.**

ArrayList of SiteSearchResults $mltSearch.execute

Code Example

```
#set($myMoreLikeThisSearch=$cms.newMoreLikeThisSearch($contentId))
$myMoreLikeThisSearch.setMinScore(60)
$myMoreLikeThisSearch.setMaxResults(10)
$myMoreLikeThisSearch.setMaxQueryTerms(5)
$myMoreLikeThisSearch.minNumCharsToBeAnalyzed(1000)
#set($results = $myMoreLikeThisSearch.execute())
#if ($results.size() == 0)
       No matching results were found for More Like This Search.<br/>
#else
       #foreach($result in $results)
              $result.score $result.title<br/>
       #end
#end
```

Template Debugging Messages

Message	Log Level	Purpose
Insufficient text to generate a More Like This Search	WARN	The content item supplied doesn't have enough characters to generate a good set of results, so an empty set of search results was returned.
Invalid content id	WARN	The content item supplied was invalid and therefore an empty set of search results was returned.
No results met the minimum search criteria	WARN	No results were found that met the configured minimum score and therefore an empty set of search results was returned.
More Like This Search was executed for the specified content id	DEBUG	Logging which content item a More Like This search was executed for.

BEST PRACTICES

Don't set the minNumCharsToBeAnalyzed too low:
Very short content items don't provide meaningful results. To get meaningful search results, you need to select a content item that has a decent amount of text to guarantee that the terms with the highest term frenquencies within the item really tell you something about the item; if they do not, the queries they generate will return a random assortment of related content. In order to prevent this, there is a configurable minimum number of characters required in the text for the supplied content item: minNumCharsToBeAnalyzed. This value should be tuned to make sure that small content items don't display with apparently random related items.

Don't set the maxQueryTerms too high:
A general rule of thumb is: as you increase this number you will get a broader set of results – however this also increases the likelihood that you will get irrelevant results. Similarly, setting this value lower will potentially return poor results as there may not be enough terms in the underlying query to find meaningful results. There is no correct number to use, but we can offer these rough guidelines:

- Value < 5 – query is too narrow to return meaningful results
- Value between 5-10 – results are mostly closely related
- Value > 10 – query is too broad so results might not all be closely related

Don't set the minTermFreq too high:
As a general guideline, don't set this value higher than 5 since it can execute a query with too few relevant terms,

and therefore generate few or no results.

Always specify addContentType or excludeContentType:
Customers often have content types that serve very different purposes. To ensure only relevant content items are returned in the 'More Like This' search, use addContentType to limit related content to one or multiple types. Alternatively, use excludeContentType to exclude one or multiple types from the results.

For media customers in particular, it is advised they exclude content types that are "Summary" or "Aggregate" format, such as a "Top Business Roundup" with headlines and summaries for the daily top business stories.

Set minWordLength to eliminate markup from results:
Although setting a minimum word length can sometimes eliminate important query terms (like "BP") it also significantly reduces the noise terms generated by HTML markup and other formatting text that often exists in content and documents.

Ad Tags

The ad tags are used to include ads served by the Clickability Platform Ad Server within the page templates.

Using Ad Tags

Preloading Ads

The information needed to display each ad on a Clickability Platform page comes from a remote ad server and is retrieved via JavaScript. In order to prevent these remote server calls from delaying page loading, a preload tag should be used that allows for the simultaneous retrieval of all the current page's ads with just one single call to the ad server.

You can use this preload capability in one of 2 ways:

1) Retrieve only those ads whose placements are used on the current page. To do this, create an arraylist containing the names of every ad placement in which an ad could appear on this page. For example:

```
#set($placements = $util.newArrayList)
#set($foo = $placements.add("Top Banner"))
#set($foo = $placements.add("Bottom Banner"))
```

or

```
#set($placements = ["Top Banner", "Bottom Banner"])
```

Then pass that arraylist into the preload tag:

```
$cms.preloadAds($placements)
```

2) Retrieve the ads for ALL Ad Placements that have been defined for your whole site (whether or not they get used on the current page). This is the default preload tag behavior if no arraylist is passed to it, so just doing:

```
$cms.preloadAds()
```

will attempt to preload an ad for every single possible ad placement. Just as with option 1) above, if no ad is targeted to that placement for that page, none will be returned; however, the process of checking every single placement might make this operation slower (for instance, if you have old Placements that you're no longer using, it's a good idea to delete them).

Checking if a Preloaded Ad was actually found for a given Placement

A further benefit of preloading your ads is that it will allow your template to check if a particular placement actually returns an ad for that page or not (allowing you to write conditional template logic like “only print the following line if there is an ad to show in the Right Sidebar placement”). Here’s how:

The preloadAds() method creates an array called plurp[] which contains a value for every placement which failed to load an ad. So you can check that array for a given placement ID, and if it’s in there, that means an ad was NOT found; if it’s *not* in there, an ad *WAS* found.

Here is an example of conditionally displaying an "Advertisement" caption and opening/closing a table row, *if and only if* an ad is found for the Top Banner placement. It assumes a“Top Banner” Ad Placement with an ID value of 5.

```
$cms.preloadAds("Top Banner")

<script language="javascript">
if(plurp && plurp[5]) {
/* No ad for placement #5 */
} else { document.write("<tr><td><span>Advertisement</span>");
}
</script>

$cms.showAd("Top Banner")

<script language="javascript">
if(plurp && plurp[5]) {
/* No ad for placement #5 */
} else {
document.write("</td></tr>");
}
</script>
```

Displaying Ads

Ad placements are referred to by name in the ad tags. For example, the following tag:

```
$cms.showAd("Top Banner")
```

includes an ad from the current website section’s Top Banner ad placement into the page.

The $cms.showAd tag outputs javascript that dynamically loads and displays the ads within the browser. This javascript is the same regardless of any of the following options available from using the Clickability Platform Ad Server:

- Image vs. Flash ads
- Single vs. Multiple ads
- Fixed size ads vs. variable sized ads

Note: You may also see a $cms.showPreloadedAd tag variant, but this tag has been deprecated, since $cms.showAd can handle displaying both preloaded and non-preloaded ads.

Displaying Ads with Additional Formatting

You may find cases where you need to specify HTML code to be displayed as part of the ad (i.e., which will ONLY display if a valid ad is found for the specified placement). This can be accomplished by using the *preHTML* and *postHTML* arguments.

For example: If you need a horizontal rule between a top-of-page Ad Placement and the element set to appear directly beneath it, but you don't know for sure that an ad will always be found for that placement (so you wouldn't want to hard-code an <hr> after the ad tag in that case), you can use:

```
$cms.showAd("Top Banner", "", "<hr color='gray'>")
<div>element beneath the ad</div>
```

and if an ad is found, it will be followed by an <hr>. Otherwise, no <hr> will be displayed. **Note:** the use of alternating quotes in the pre/postHTML argument values. **Also note:** If you want to use a Velocity expression as part of your pre/postHTML arguments, make sure to enclose the parameter within DOUBLE quotes, as single quotes do not get Velocity-parsed.

Note: For Text Ads, passing the preHTML and postHTML arguments into the ad tag is not recommended. See Working with Text Ads section below.

Displaying a Particular Ad

If it is desired to place a specific ad within a page template (vs. letting the ad selection rules choose from among all ads targeted to a given placement), the ad's ID number can be included in the page template. This will override all standard rules of ad selection from the ad server and display the specific ad referenced in the tag.

No Script Option

The javascript output from the $cms.showAd tags do not include a <noscript> option by default. This means that the default behavior for a browser with javascript disabled is to ignore the ad tag, potentially causing formatting issues.

If the show NoScript option is set to true, the ad tag created within the page will include a <noscript> option that places a blank image of the ad placement's size within the page, thus preserving the formatting and page layout of the page.

Working with Text Ads

Because Text Ads can potentially be more complicated to serve (e.g. ones that contain JavaScript), there are some best practices of using ad tags that should be used with placements that may display Text Ads.

Pre/Post HTML

For ad placements that may show a Text Ad, it's best not to pass preHTML and postHTML arguments into the $cms.showAd() tag. If it is also not ideal to hard-code the conditional HTML into the page (since you only want them to show if an ad is actually returned for that placement), there is another solution. The solution will work *ONLY if you've preloaded the ad placement*. If you do this, you can use javascript to check for an ad before including the desired preHTML and postHTML. See "Checking if a Preloaded Ad was actually found for a given Placement" in this chapter for details on how to do that.

Special Text Ad parameters

If a third-party ad tag for which you're creating a Text Ad requires that a random number or timestamp be dynamically inserted into the ad, simply put `{random}` and/or `{timestamp}` into the Text Ad where called for, and the Dynamic Site Platform ad server will replace the values accordingly before serving the ad.

Tracking Text Ad Clicks

Since a Text Ad can contain any number and placement of links, standard tracking does not apply, which means Text Ads are NOT included in the standard WCM Advertisement Reports. One alternative is to use $imware.trackedURL within the Text Ad, which will appear in the WCM Marketing Reports for "Offsite Link Clicks". See the "URL Tracking" section of the "Tracking" chapter for the syntax of $imware.trackedURL.

Flash Ad Tracking

Flash ads are displayed in the same manner as other ads in the Dynamic Site Platform and nothing unique needs to be done to include them on the website or to track their views.

However, to track the clickthroughs on Flash ads, requires special coding within the Flash file itself. The Flash button script code should include the following function exactly:

```
on(release) {
               if(clickTAG.substring(0, 4)=="http") {
                getURL(clickTAG, "_blank");
               }
            }
```

A clickTAG parameter is automatically passed to the Flash file by the ad server. This clickTAG is actually a redirection URL to the final target URL of the ad as specified in the Clickability Platform. The test for http is some minimal protection that the link is a valid URL.

Limelight Video Portal (LVP) Integration

LVPContext Object	$cms.getLVPContext()
Given a LVPContext object $lvpContext:	
	$lvpContext.setEmbedType() – used to set one of the three embed-code styles. The input is a string; in order to get the appropriate string value use one of these methods: getWebSiteEmbedType(), getBlogEmbedType() or getLinkEmbedType()
	$lvpContext.setCustomFlashVar() – an advanced capability exposed to the user to customize player behavior
	$lvpContext.setMediaId() – used to set the LVP media ID of the asset which will be loaded and played
	$lvpContext.setChannelId() – used to set a channel that should be loaded in the player
	$lvpContext.setChannelListId() – used to load and display channel groups in the player
	$lvpContext.setPlayerForm() – used to set the player that should be used to display the asset. The tag expects the name of the player that should be used; to get a list of all the players available navigate to Channels > Player Builders in the LVP interface
	$lvpContext.setWidth() – used to override the width of the player that is set in the LVP interface
	$lvpContext.setHeight() – used to override the height of the player that is set in the LVP interface
	$lvpContext.embedCode – generates the embed code
	$lvpContext.channelMetaData – returns an array of LVP channels
Given a LVPChannels object $lvpChannels:	
String	$lvpChannels.id
String	$lvpChannels.description
String	$lvpChannels.title
Boolean	$lvpChannels.emailEnabled
Boolean	$lvpChannels.embedEnabled
Boolean	$lvpChannels.searchInsideEnabled
Boolean	$lvpChannels.autoplayEnabled
Boolean	$lvpChannels.rssEnabled
Boolean	$lvpChannels.published
	$lvpContext.getPopularVideos(string startDate, string endDate) – uses LVP analytics to output Most Popular Videos in JSON format
	$lvpContext.getPopularVideos(string startDate, string endDate, int maxRecords)

$lvpContext.getPopularVideos(string mediaId, string startDate, string endDate, int maxRecords)

Usage:

- **startDate** – start of Most Popular date range; format dates as MM/dd/yyyy
- **endDate** – end of Most Popular date range; format dates as MM/dd/yyyy
- **maxRecords** – maximum number of videos to return
- **mediaID** – ID of the media asset in LVP

LVP Code Samples

Generate sample embed code for a content detail page:

```
#set($lvpId=$cms.content.extID)
#set($lvpC = $cms.getLVPContext())
$lvpC.setEmbedType($lvpC.getWebSiteEmbedType())
$lvpC.setPlayerForm('LVPPlayer')
$lvpC.setMediaId($lvpId)
$lvpC.setCustomFlashVar('disableBitrateSelection=true')
Embed Code: $lvpC.getEmbedCode()
```

Get content representation of LVP media asset using “Related Content” feature on a content detail page:

```
#set($item = $cms.content)
<table width="100%">
<tr><td colspan="2"><b>$item.title </b></td></tr>
<tr><td width="35%">
       #set($lvpId='')
       #foreach($content in $item.linkedContent('videoAssociation'))
              #set($contentItem = true)
              #set($ibContent = $cms.content($content, true))
              #if($velocityCount == 1)
                     <strong>RELATED ITEMS (Item Browser):</strong><br>
              #end
              #if($ibContent)
                     $ibContent.id - $!ibContent.title</a><br>
                     #set($lvpId=$ibContent.extID)
                     #set($ibContent = false)
              #end
       #end
       #set($lvpC = $cms.getLVPContext())
       $lvpC.setEmbedType($lvpC.getWebSiteEmbedType())
       $lvpC.setPlayerForm('LVPPlayer')
       $lvpC.setMediaId($lvpId)
       $lvpC.setCustomFlashVar('disableBitrateSelection=true')
       Embed Code: $lvpC.getEmbedCode()
</td><td>
       $item.body
</td>
</tr>
</table>
```

Output each LVP channel and properties:

```
#set($lvpC = $cms.getLVPContext())
#foreach($channel in $lvpC.getChannelMetaData())
       $channel.lvpChannelId <br />
       $channel.description <br />
       $channel.title <br />
       $channel.emailEnabled <br />
       $channel.embedEnabled <br />
       $channel.searchInsideEnabled <br />
       $channel.autoplayEnabled <br />
       $channel.rssEnabled <br />
       $channel.published     <br />
#end

#foreach($video in $cms.filteredLinks("Video Content Item", 1, 10))
#set($lvpC = $cms.getLVPContext())
#foreach($channelID in $lvpC.getChannelsForMedia($video.id))
       LVP Channel ID: $channelID <br />
#end

Note in that example, $video.id is the Clickability Platform content ID and
$channelID is the channel ID from LVP
```

Code to output Most Popular LVP videos for a particular date range:

```
#set($lvpC = $cms.getLVPContext())
$lvpC.getPopularVideos("03/01/2012", "04/23/2012")
```

Third Party Integrations

In addition to Clickability Platform functionality, we have developed tools for integration with key third party vendors. This chapter will outline tools for:

- Inform
- Amazon SimpleDB & Amazon RDS
- SOAP Messaging
- Template-based JSON Parser

Inform

Inform Articles Specific to Customer Account:

String List	**$inform.getTopSubjectsForCustomer**(int interval, char intervalType[, int destID], int numberOfSubjects, int customerId[, date endDate])
String List	**$inform.getSubjectForRelatedArticlesByContentId**(int contentId, int score, int numberOfSubjects, int customerId[, boolean applyTopic])
Object list	**$inform.getContentInformationForRelatedArticles**(int contentId, int score, int numberOfSubjects, int customerId, int numberofRelatedArticles[, int interval, char intervalType][, int destID])
Object List	**$inform.relatedCmsContentForSubject**(string subject, int customerId, int domainId, int maxNumberOfRecords[, int score])

Usage:

- **interval** – Period of time content was targeted or created within the following period calculated from current date. To be used in conjuction with intervalType.
- **intervalType** – Type of time to be used with interval. Can be any one of the following: D for day, M for month and W for week.
- **destID** – the destID where the content is targeted. The default is all.
- **domainId** – the domain ID where the content is located.
- **numberOfSubjects** – The number of subjects to be considered for associated content items.
- **customerId** – The customer account ID from which items are to be retrieved.
- **endDate** – The parameter is optional to specify a different date range. It will include all content before endDate but after (endDate – interval intervalType).
- **contentId** – content ID for which related articles should be retrieved
- **score** – the threshold for subject relevancy (1-100)
- **numberofRelatedArticles** – The number of items to be pulled back.
- **customerId** – The customer account ID from which items are to be retrieved.
- **applyTopic** – when this is passed as true, it will restrict the results to subjects that are categorized by Inform as TOPICS. Default is false.
- **numberofRelatedArticles** – the number of content items to be returned in the result
- **maxNumberOfRecords** – max number of items to be returned

Given an InformRelatedContentItems object $informContent:

Object List	$informContent.articles
Object List	$informContent.audio
Object List	$informContent.blogs
Object List	$informContent.video

Given an InformRelatedContentItem object $informContentItem:

$informContentItem.attribute-name

Attributes:

- headline
- title
- date
- url
- blurb
- publication
- score

More information about Inform can be found here: http://www.informpublisherservices.com/. For more information about how to setup an account with Inform, please contact your account representative.

AWS SimpleDB

Amazon is a third-party application that we have provided integration tools for. More information about Amazon Web Services SimpleDB can be found here: http://aws.amazon.com/simpledb/.

Terminology:

- DomainName in Simple DB is equivalent to a table name.
- IdentifyingKey in Simple DB is equivalent to a primary key.
- AttributeName in Simple DB is equivalent to a column name.
- AttributeValue in Simple DB is equivalent to a column value.

	$simpleDB.setCredentials(string awsId, string accessKey)
Object	$simpleDB.query(string query)
Object	**$simpleDB.addAttribute**(string name, string value, boolean replace)
Boolean	**$simpleDB.update**(arraylist attributes, string domainName, string identifyingKey)
Boolean	$simpleDB.delete(string identifyingKey, string domainName)

Given a SimpleDBResponse object $resp:

Boolean	$resp.isSetSelectResult()
Object	$resp.selectResult

Usage:

- **awsId, accessKey** – The Amazon instance ID and access key
- **query** – a SQL query
- **name** – attributeName
- **value** – attributeValue
- **replace** – 'true' to replace existing value and 'false' to add a new value
- **attributes** – an arrayList of replaceable attributes
- **domainName** – the table name
- **identifyingKey** – the primary key

Examples:

$simpleDB.query(string *query*)

Query the database for specific values. This method will return a SimpleDBResponse object. Use the available methods to iterate over the result set.

```
#set($resp = $simpleDB.query('select * from Clickability'))
#set ($items = $resp.selectResult.item)
#foreach($item in $items)
              #if ($item.isSetName())
                     $item.name
              #end
              #set ($attributes = $item.attribute)
              #foreach ($attribute in $attributes)
                     #if ($attribute.isSetName())
                           $attribute.name
                     #end
                     #if ($attribute.isSetValue())
                           $attribute.value
                     #end
              #end
#end
```

$simpleDB.addAttribute(string *name*, string *value*, boolean *replace*)

In order to insert/update data in a domain, users need to define the attributeName and attributeValue pairs and pass these to the update method on $simpleDB object. The 'replace' parameter of the method, identifies if the pre-existing value of the attribute will be replaced or the new value will be added to the record.

```
#set($attributeList = $util.newArrayList)
#set($foo =
$attributeList.add($simpleDB.addAttribute('CustomerName','NewCustomer',true)
)
```

$simpleDB.update(arraylist *attributes*, string *domainName*, string *identifyingKey*)

Using the attribute arrayList created in the previous method, use this method to insert or update the table. If the operation was performed successfully, boolean true is returned, else false.

```
$simpleDB.update($attributeList,'Clickability', '8500100')
```

$simpleDB.delete(string *identifyingKey,* string *domainName*)

This method allows users to delete records identified by the identifyingKey in the domain. This method will return boolean true if the operation was performed successfully, else false.

```
$simpleDB.delete('8500100', 'Clickability')
```

Amazon RDS

RDS is a relational database service exposed by Amazon that allows end users to host a MySQL db instance as a web service. For more information, see: http://aws.amazon.com/rds/. The purpose of this document is to illustrate the implementation in platform to access this service and data from an amazon hosted external database.

$amazonRDS.configureDB(string <RDS Server End Point/Server Name>, string Port, string User, string Password, string <schemaName/DB Name>)

ArrayList	$amazonRDS.executeQuery(string SQL query)
Int	$amazonRDS.executeInsert(string SQL query)
Int	$amazonRDS.executeDelete(string SQL query)
Int	$amazonRDS.executeUpdate(string SQL query)

Examples:

$amazonRDS.executeQuery(string *SQL query*)

This method allows users to query the RDS db instance's schema that was specified in the 'configureDB' method. The SQL must be valid for it to be executed on RDS DB. The SQL can have filters, subqueries, joins etc. This method will return an ArrayList of a Map object. Each map has data related to one row returned from the query. You have 2 ways of accessing the result data from the map:

```
- map.get('<tableName>.<columnName>') or
- map.get('<columnName>')
```

In cases where invalid SQL is passed and results in exception or validation fails, a null value is returned from the tag. If no data is returned from a valid query, an ArrayList object will be sent back but will be empty.

```
#set($foo=$amazonRDS.configureDB('clickrds.rds.amazonaws.com', '3306',
'user', 'password', 'rdsdemo'))
#set($results=$amazonRDS.executeQuery('SELECT name FROM demo'))
<b>Names Fetched from RDS</b>
<br/>
#foreach($result in $results)
   $result.get('demo.name')
   <br/>
#end
```

$amazonRDS.executeInsert(string *SQL query*)
$amazonRDS.executeDelete(string *SQL query*)
$amazonRDS.executeUpdate(string *SQL query*)

The return type of these methods is an int, indicating the count of records that were affected by the SQL on the DB. These methods will return -1 in cases where there is an exception caused by the statement or validation fails.

```
#set($sql="insert into demo (name) values('ClickTest')")
#set($resultInt = $amazonRDS.executeInsert($sql))
```

Restricted Schema Names

The following is a list of invalid schema names that have been reserved (case-insensitive).

- CMS
- Stats
- Repo
- Report
- Pub

SOAP Messaging from Templates

This feature will allow you to construct a SOAP message to an outside service and receive a response that can then be parsed within a template.

SOAP, ("Simple Object Access Protocol") is a commonly used Web Services protocol, similar in usage to other protocols such as REST, XML-RPC, and JSON-RPC. As a Web Service, SOAP enables individuals to message an external service with remote procedure calls that return XML-formatted messages to be interpreted by the messenger. The types of calls that can be made via SOAP are described in a WSDL ("Web Services Description Language") document, which contains the set of exposed Web Service methods available to interact with.

There are several versions of the SOAP protocol, and further, several different bindings that are available in a WSDL depending on the type of message expected by the external service (e.g. RPC/encoded, Document/Literal, etc), all of which affect the formatting of the messages.

Given the potential complexity of the above, the expectation is that any customer utilizing the SOAP messaging utilities has staff trained on Web Services and constructing SOAP messages.

SOAPMessage	$soap.newMessage() Creates a new SOAPMessage object (SOAP 1.1) with the default SOAPPart, SOAPEnvelope, SOAPBody, and SOAPHeader objects.
SOAPMessage	$soap.new12Message() Creates a new SOAPMessage object (SOAP 1.2) with the default SOAPPart, SOAPEnvelope, SOAPBody, and SOAPHeader objects. **$soap.addNamespaceToEnvelope**(SOAPMessage msg, String prefix, String uri) Adds a namespace of prefix:uri in the SOAPEnvelop of this SOAPMessage. **$soap.addSOAPHeaderElement**(SOAPMessage msg, String headerElementString) Creates a element from the headerElementString and adds it to the SOAPHeader of this SOAPMessage. **$soap.addSOAPBodyElement**(SOAPMessage msg, String bodyElementString) Creates a element from the bodyElementString and adds it to the SOAPBody of this SOAPMessage.
SoapMessage	**$soap.sendmySoapRequest**(String wsdlURL, String namespaceURI, String serviceName, String portName, SOAPMessage msg, String soapAction) Sends this SOAPMessage to the service specified in the wsdl at wsdlURL, identified by serviceName at portName with this namespaceURI. It is recommended that the soapAction is supplied, but it's not required. If it is not provided, an attempt will be made to retrieve the soapAction from the specified wsdl.
SOAPMessage	$soap.stringToSOAP(String soapText) Creates a new SOAPMessage object (SOAP 1.1) from this soapText string.
SOAPMessage	$soap.stringToSOAP12(String soapText) Creates a new SOAPMessage object (SOAP 1.2) from this soapText string.
String	$soap.soapToString(SOAPMessage msg)

	Convert this SOAPMessage to a string of xml.
SOAPMessage	**$soap.documentToSOAP**(Document soapDocument) Creates a new SOAPMessage object (SOAP 1.1) from this soapDocument.
SOAPMessage	**$soap.documentToSOAP12**(Document soapDocument) Creates a new SOAPMessage object (SOAP 1.2) from this soapDocument.
String	$soap.soapToDocument(SOAPMessage msg) Convert this SOAPMessage to an xml Document.

Constructing SOAP Messages

SOAP messages can be constructed in Velocity through several different methods:

- Messages can be written as a string (or combination of concatenated strings) in Velocity. This is generally useful for shorter, less complicated messages
- Messages can be constructed piecemeal via a SOAP message constructor utility available in through the Velocity templates. This is generally useful for longer, more complex messages

Note: Currently, SOAP messages that include attachments are not supported.

The construction of a SOAP message (along with the delivery of this message) is done within the Velocity context, $soap.

When constructing your messages as a string object, the string can be converted to a SOAP message using the $soap method $soap.stringToSOAP (which creates a SOAP 1.1 object). Calling $soap.stringToSOAP should be saved as an object that can then be delivered to the Web Service in question. Example,

```
#set($xmlStr='<?xml version="1.0" encoding="UTF-8"?>')
#set($envStr='[YOUR SOAP ENVELOPE STRING]')
#set($headStr='[YOUR SOAP HEADER STRING]')
#set($bodyStr='[YOUR SOAP BODY STRING]')
#set($reqStr="$envStr$headStr$bodyStr")
#set($soapreq=$soap.stringToSOAP($reqStr))
```

Similarly, SOAP 1.2 objects can be made with, $soap.stringToSOAP12
Otherwise, constructing the SOAP message by its components by constructing a new SOAP message object using the call, $soap.newSOAPMessage (this will instance a SOAP 1.1 object). An object should be made with this call that you can then perform several actions on:

- Name spaces can be added to the envelope
- Header Elements can be added
- Body Elements can be added

Similarly, SOAP 1.2 objects can be made with $soap.new12Message

Each portion of the message is written as a string, and can then be inserted into SOAP message object via a variety of methods, $soap.AddNamespaceToEnvelope, $soap.addSOAPHeaderElement, and $soap.addSOAPBodyElement. Once all of the necessary components have been added, the object is then ready to be delivered to the Web Service. Example (using the Clickability Platform API),

```
#set($soapReq=$soap.NewSOAPMessage)
$soap.addNamespaceToEnvelope($soapReq, "cred",
"http://credential.transport.v1.api.cmpublish.clickability.com")
$soap.addNamespaceToEnvelope($soapReq, "tran",
"http://transport.v1.api.cmpublish.clickability.com")
#set($headerStr='<cred:credentials username="[YOUR USERNAME]"
password="[YOUR PASSWORD]" customerID="[YOUR CUSTOMER ID]"/>')
$soap.addSOAPHeaderElement($soapReq, $headerStr)
#set($bodyStr='[YOUR BODY STRING]')
$soap.addSOAPBodyElement($soapReq, $bodyStr)
```

Delivering SOAP Messages

Once a SOAP message has been constructed, a method is available through Velocity that sends the message out to the external service. In order to deliver the message, the following needs to be known:

- The URL for the WSDL
- The target namespace for the service (URL)
- The service name
- The port name
- The message itself (constructed earlier)

The result of the call, $soap.sendSOAPRequest, should be made into a new object that can be parsed by the template. **Example,**

```
#set($response=$soap.sendSOAPRequest('http://cms.clickability.com/api/API?ws
dl','http://transport.v1.api.cmpublish.clickability.com', 'API',
'APIHttpPort', $soapReq))
```

where $soapReq represents the SOAP message as constructed with either of the methods mentioned above.

Parsing SOAP Messages

Once the object is available, the response can be handled in three ways:

1. The output can be converted to a string
2. The output can be converted to an XML document
3. The output can be parsed using an XML node querying language called, “XPath”

Note: Currently, SOAP responses that return attachments are not supported.

The $xpath context contains a variety of methods to help navigate the contents of the SOAP message as a tree using XPath expressions. These methods include,

- `$xpath.getString` – takes an XPath expression and the SOAP object and returns the result to a string
- `$xpath.getNumber` – takes an XPath expression and the SOAP object, and returns a double
- `$xpath.getNode` – takes an XPath expression and the SOAP object and converts the result to an XPath Node
- `$xpath.getNodeList` – takes an XPath expression and the SOAP object and converts the results to a list of XPath Nodes

In case of an error, the SOAP message can be checked for faults by pulling back the body of the message and

inspecting it for faults. The message can be inspected for faults by calling `.getSOAPBody().hasFault()` on an object, which returns a Boolean. If there is a fault, an object can be made containing the fault code and message, which is then retrieved with .faultCode and .faultString. Example,

```
#if($response.getSOAPBody().hasFault())
#set($fault=$response.getSOAPBody().getFault())
faultcode: $fault.faultCode
faultstring: $fault.faultString
#end
```

Examples:

The following snippets are examples from publicly accessible web services:

Quote of the Day Example:

```
<div>
##set request all at once
##set($xmlStr='<?xml version="1.0" encoding="UTF-8"?>')
#set($envStr='<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:swan="http://swanandmokashi.com">')
#set($bodyStr='<soapenv:Body><swan:GetQuote/></soapenv:Body></soapenv:Envelo
pe>')
#set($reqStr="$envStr$bodyStr")
#set($soapreq=$soap.stringToSOAP($reqStr))

#set($response=$soap.sendSOAPRequest('http://www.swanandmokashi.com/HomePage
/WebServices/QuoteOfTheDay.asmx?WSDL', 'http://swanandmokashi.com',
'QuoteofTheDay', 'QuoteofTheDaySoap', $soapreq))
#set($respStr=$soap.soapToString($response))

#*
#set($reqStr=$soap.indentAndEscapeHtml($reqStr))
<h4>request</h4>
<pre>${reqStr}</pre>

#set($respStr=$soap.indentAndEscapeHtml($respStr))
<h4>response</h4>
<pre>${respStr}</pre>
*#

##test xpath
<h2>Quote of the day</h2>
#if($soap.hasFault($response))
<font color=red>Problem retrieving quote of the day: </font>
$soap.getFaultString($response)
#elseif($xpath.hasNode("//GetQuoteResult", $response))
<h4>Quote of the day:</h4>
$xpath.getString("//QuoteOfTheDay", $response)
<h4>Author:</h4>
$xpath.getString("//Author", $response)
#else
<h4>Quote of the day service is not available</h4>
#end

</div>
```

Weather Service Example:

```
<div>
##<h2>set request all at once</h2>
#set($envStr='<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://www.webservicex.net">')
#set($bodyStr='<soapenv:Body><web:GetWeatherByZipCode><web:ZipCode>94104</we
b:ZipCode></web:GetWeatherByZipCode></soapenv:Body></soapenv:Envelope>')
#set($reqStr="$envStr$bodyStr")
#set($soapReq=$soap.stringToSOAP($reqStr))

##send request, also pass the soapAction as the last parameter
#set($response=$soap.sendSOAPRequest('http://www.webservicex.net/WeatherFore
cast.asmx?WSDL', 'http://www.webservicex.net', 'WeatherForecast',
'WeatherForecastSoap', $soapReq,
'http://www.webservicex.net/GetWeatherByZipCode'))
#set($respStr=$soap.soapToString($response))

#*
#set($reqStr=$soap.indentAndEscapeHtml($reqStr))
<h4>request</h4>
<pre>${reqStr}</pre>

#set($respStr=$soap.indentAndEscapeHtml($respStr))
<h4>response</h4>
<pre>${respStr}</pre>
*#

##test using xpath combined with DOM API
<h4>Local Weather</h4>
#if($soap.hasFault($response))
<font color=red>Problem retrieving weather info: </font>
$soap.getFaultString($response)

#elseif($xpath.hasNode("//Details", $response))
##construct a table of weather forecast
#set($place=$xpath.getString("//PlaceName", $response))
#set($state=$xpath.getString("//StateCode", $response))
#set($weatherNodes=$xpath.getNodeList("//WeatherData", $response))
#set($count=$weatherNodes.length)

<b>$place, $state</b>
<table border=1>
<tr><th>Day</th><th>Weather</th><th>High</th><th>Low</th></tr>
#foreach($i in [1..$count])
#set($j = $i - 1)
#set($curWeatherNode=$weatherNodes.item($j))
#if($curWeatherNode.hasChildNodes())
#set($childrenNodeList=$curWeatherNode.childNodes)
#set($dayNode=$childrenNodeList.item(0))
#set($imageNode=$childrenNodeList.item(1))
#set($highNode=$childrenNodeList.item(2))
#set($lowNode=$childrenNodeList.item(3))
<tr>
<td>$dayNode.firstChild.textContent</td>
<td><img src="$imageNode.firstChild.textContent"/></td>
<td>$highNode.firstChild.textContent</td>
<td>$lowNode.firstChild.textContent</td>
</tr>
#end
#end
</table>
```

```
#else
<h4>Weather info not available</h4>
#end
</div>

#*
##test using xpath only
<h4>Local Weather</h4>
#if($soap.hasFault($response))
<font color=red>Problem retrieving weather info: </font>
$soap.getFaultString($response)

#elseif($xpath.hasNode("//Details", $response))
##construct a table of weather forecast
#set($place=$xpath.getString("//PlaceName", $response))
#set($state=$xpath.getString("//StateCode", $response))
#set($weatherNodes=$xpath.getNodeList("//WeatherData", $response))
#set($count=$weatherNodes.length)

<b>$place, $state</b>
<table border=1>
<tr><th>Day</th><th>Weather</th><th>High</th><th>Low</th></tr>
#foreach($i in [1..$count])
#set($exp="//WeatherData[${i}]")
exp=$exp
#set($curWeatherNode=$xpath.getNode($exp, $response))
#set($day=$xpath.getString("Day", $curWeatherNode))
#set($image=$xpath.getString("WeatherImage", $curWeatherNode))
#set($high=$xpath.getString("MaxTemperatureF", $curWeatherNode))
#set($low=$xpath.getString("MinTemperatureF", $curWeatherNode))
<tr>
<td>$day</td>
<td><img src="$image"/></td>
<td>$high</td>
<td>$low</td>
</tr>
#end
</table>

#else
<h4>Weather info not available</h4>
#end
</div>
*#
```

Template-based JSON Parser

To facilitate integrations with third party tools that require working with JSON objects through Velocity, there are several methods available in the $jsonParser context. The objects referenced below conform to the specifications available here,
http://www.json.org/javadoc/index.html?org/json/JSONObject.html
http://www.json.org/javadoc/index.html?org/json/JSONArray.html
with the exclusion of write methods. Note the details below describe a sample use-case rather than the full use.

```
$jsonParser.createJSONObject()
```

Used to instance new JSON objects in Velocity. An object has to be created prior to inserting data into it. Once you've instanced an object, methods like `$object.put(String key, object value)` and `$object.get(String key)` can be used to retrieve data.

$jsonParser.createJSONArray()
Used to instance new JSON Array objects in Velocity. An object has to be created prior to inserting data into it. Once the object has been instanced, methods such as `$object.put(int index, object value)` and `$object.get(int index)` can be used to get and retrieve data.

```
$jsonParser.parseJSON(<JSON String>)
```

Allows a user to create a new JSON object by passing a JSON string retrieved from a third party service.

```
$jsonParser.parseJSONArray(<JSON Array String>)
```

Allows a user to create a new JSON array object by passing a JSON Array as a string.

```
$jsonParser.getJSONString(JSONObject)
```

Converts $jsonParser objects to a string value to be sent to a web service.

To use the parser, first a new JSON object needs to be instanced. Once it's been created, an object can be added to it: either a literal (number, string, boolean), another JSON object, or a JSON array, by way of .put() methods.

After the object has been made and populated, it can be converted to a string to be sent to a web service via the getJSONString() method.

Sample usage:

```
#set($jsonObj = $jsonParser.createJSONObject())
#set($empty = $jsonObj.put("test1", "Hello World"))
Output of jsonObj: $jsonParser.getJSONString($jsonObj)
<br/>
Value: $jsonObj.getInt("invalid key")
<br/>
#set($jsonString = "{elementId : newBtn, isDelegation: true}")
#set($jsonObj = $jsonParser.parseJSON($jsonString))
Output of jsonObj: $jsonParser.getJSONString($jsonObj)
```

Web Marketing Acceleration (WMA)

WMA Reference Materials

There are several documents available on the User Community for WMA which will be needed during any WMA implementation. These are located at:
http://community.clickability.com > Knowledge Base > WMA.

WMA Configuration

These tags provide a quick way to retrieve information about the WMA WCM configuration. Further details are in the sections below.

Configuration	
ArrayList of Action Definitions	$wma.actionDefinitions
Action Definition Object	$wma.getActionDefinition(string actionTag)
ArrayList of Interest Definitions	$wma.interestDefinitions
Interest Definition Object	$wma.getInterestDefinition(string interestTag)
ArrayList of Visitor States	$wma.stateDefinitions
ArrayList of PPL Fields	$wma.getProfileFieldsForLevel(string profileLevelTag)
ArrayList of Profile Field Objects	$wma.profileFields
Profile Field Object	$wma.getProfileField(string profileTag)
ArrayList of PPL	$wma.profileLevels
PPL Object	$wma.getProfileLevel(string profileLevelTag)

Notes:

- **actionTag** – Velocity tag name of Action
- **interestTag** – Velocity tag name of Interest

Progressive Profiles

WMA Progressive Profile Levels (PPL) are configurable in the CMS. Each progressive profile level contains information obtained from a website visitor via a Progressive Profile Form. When the visitor provides the requested information (email address, phone number, job title, company, etc), the visitor is moved to the designated PPL.

Note: as a best practice, always ask for an email address in order to merge anonymous cookies with visitor data.

The progressive profile form website section is configured in the WMA tab of the domain detail page: Admin tab > Design > Domains.

The website section can be any section on the domain. Specify the url with a leading "/" like:

/profileForm
/templates/profile_template

The template developer needs to set up a website section or set up a translated template to host the page. When the profile page is rendered, the template developer will be able to use a collection of template tags to dynamically present the desired form to the website visitor. In particular, the following tags have been created:

- `hasGatedContent()`
- `getGatedContent()` -- Access the protected content item which the visitor attempted to get. Perhaps to display the content title, etc.
- `getProfileLevelForGatedContent()`
- `getUnsetFieldsForGatedContent()` -- Determines the fields necessary to attain the require PPL

The profile form MUST be POSTED to the "/s" path with the form action parameter value set to "action=submitProfile".

Example:

```
<form name="profileForm" method="post" action="/s">
<input type="hidden" name="action" value="submitProfile" />
<!-- profile field inputs, etc. -->
</form>
```

The "submitProfile" function will store all parameters corresponding to progressive profile fields to the VMS server. The profile fields must be submitted in a very specific format.

If the submitted data is sufficient to access the desired content item:

- The PPL will be granted to the visitor.
- The system will track that access to the content item was granted
- The visitor will be redirected to the originally requested content item.

If the submitted data is insufficient and the PPL is not attained:

- The visitor will be redirected back to the Progressive Profile Form website section.
- The Progressive Profile Form template will maintain the values that were submitted and prompt the user for the remaining information.
- Any submission errors will be available to the template developer.

Progressive Profile Page	
Boolean	$wma.isProfilePage() Returns true if the current website section path is the same as that defined as the the progressive profile form website section via the Admin tab > Design > Domains > WMA.

Progressive Profile Level	
ArrayList of PPL Objects	$wma.profileLevels
PPL Object	$wma.getProfileLevel(string profileLevelTag)
PPL Obejct	$wma.profileLevel

Notes:

- **profileLevelTag** – Velocity tag name for PPL
- **profileTag** – Velocity tag name for the profile field

Given a Progressive Profile Level object $progressiveProfileLevel:

String	$progressiveProfileLevel.description
String	$progressiveProfileLevel.name
Integer	$progressiveProfileLevel.level
Integer	$progressiveProfileLevel.scoreTrigger
String	$progressiveProfileLevel.toString()
String	$progressiveProfileLevel.tag
ArrayList of Profile Field Objects	$progressiveProfileLevel.fields

Profile Fields

ArrayList of Profile Field Objects	$wma.profileFields
ArrayList of Profile Field Objects	$wma.getProfileFieldsForLevel(string profileLevelTag)
ArrayList of Profile Field Objects	**$wma.getUnsetFieldsForProfileLevel**(string profileLevelTag) – automatically returns all profile fields that have not yet been submitted for specified profile level; note when used with segmentation, this tag incorporates rules and shows the rule applied field(s) only when the rule evaluates to true
ArrayList of Profile Field Objects	**$wma.getUnsetFieldsForProfileLevel**(integer profileLevel) – automatically returns all profile fields that have not yet been submitted for specified profile level; note when used with segmentation, this tag incorporates rules and shows the rule applied field(s) only when the rule evaluates to true
Profile Field Objects	$wma.getProfileField(string profileTag)

Notes:

- **profileLevelTag** – Velocity tag name for PPL
- **profileLevel** – number of PPL
- **profileTag** – Velocity tag name for the profile field

Given a Profile Field object $profileField:

String	$profileField.dataType
String	$profileField.defaultValue
String	$profileField.fieldType
String	$profileField.label
Integer	$profileField.leadScoreContribution
Integer	$profileField.progressiveProfileLevel
String	$profileField.tag

Integer	$profileField.numberOfLines
Integer	$profileField.maxLength
Integer	$profileField.minLength
ArrayList of Select Field Options	$profileField.options
dataType Type	$profileField.value
String	$profileField.toString()

Notes:

- **dataType options** – text, longText, number, date, dateTime, email, boolean
- **fieldType options** – text, textArea, select, multiSelect
- **numberOfLines** – number of lines in textArea field type
- **maxLength** – input max length for text field type
- **minLength** – input min length for text field type

Given Select Field options $selectOption:

Integer	$selectOption.value
String	$selectOption.label

Segmentation

Segmentation enables marketers to ask different progressive profile questions based on previously submitted profile answers. The following tags provide access to the "rules" configured in the WMA interface for the purpose of segmentation.

ArrayList of RuleDefinitions	$wma.ruleDefinition Returns all the active rule definitions.
ArrayList of RuleDefinitions	$wma.getRuleDefinition(integer *ruleDefinitionId*)
ArrayList of RuleDefinitions	$wma.getRuleDefinitionByName(string *ruleName*)

Notes:

- **ruleDefinitionId** – ID of the rule defined via the WMA tab > Administration > Rules interface
- **ruleName** - Name of the rule defined via the WMA tab > Administration > Rules interface

Given a RuleDefinition object $ruleDefinition:

Integer	$ruleDefinition.ruleId
String	$ruleDefinition.ruleName
Logical Operator	$ruleDefinition.logicalOperator
Integer	$ruleDefinition.customerId
ArrayList of RuleExpression Objects	$ruleDefinition.ruleExpressions

Given a RuleExpression object $ruleExpression:	
Integer	$ruleExpression.fieldId
String	$ruleExpression.fieldName
Logical Operator	$ruleExpression.expressionOperator
ArrayList of RuleExpressionValue Objects	$ruleExpression.ruleExpressionValues
Given a RuleExpressionValue object $ruleExpressionValue:	
String	$ruleExpressionValue.ruleName
String	$ruleExpressionValue.ruleValueLabel

Visitor Data

The following tags retrieve data about the website visitor. Use these tags to dynamically display content to the visitor.

Basic Visitor Data

Profile Object	$wma.profile
Progressive Profile Level Object	$wma.profileLevel
Integer	$wma.leadScore
ArrayList of Lead Score Objects	$wma.leadScoreAdjustments
Integer	$wma.getInterestScore(string *interestTag*)
Integer	$wma.getVisitorActionCount(string *actionTag*)
Integer	$wma.pageViewCount
ArrayList of Page View Objects	$wma.pageViews
Integer	$wma.searchCount
Integer	$wma.visitCount
Boolean	**$wma.isMarketingActivityBlocked**() – **Note:** this flag is intended to be set via the API so an external Marketing Automation or CRM tool stops further nurturing
Date	$wma.firstActivity
Date	$wma.lastVisitDate
String	$wma.state
Boolean	$wma.isLoggedIn()
Boolean	$wma.isRegistered()
String	$wma.timeZone
String	$wma.profile.getField(string *profileTag*)

Notes:

- **interestTag** – Velocity tag name for Interest
- **actionTag** – Velocity tag name for Action
- **profileTag** – Velocity tag name for Action

Visitor Resource Portfolio

ArrayList of Portfolio Objects	$wma.getContentInPortfolio(boolean isAddedByCmsUser)
Boolean	**$wma.isContentInPortfolio**(boolean isAddedByCmsUser, int destID, int contentid)
String *	$wma.getAddContentToPortfolioUrl(int destID, int contentID)
String *	$wma.getRemoveContentFromPortfolioUrl(int destID, int contentID)

Notes:

- **isAddedByCmsUser** – true if added by WCM admin, false if added by website visitor
- **destID** – the website section destination ID where the content item is targeted; this is NULL when isAddedByCmsUser is true
- **contentID** – the content ID of the item to be added or removed; items cannot be removed when isAddedByCmsUser is true

* These tags create URLs that, when clicked, will add/remove items from the resource portfolio. Items added by WCM admins cannot be removed by the website visitor.

Visitor Content Gating

Boolean	$wma.hasGatedContent()
Content Object	$wma.gatedContent
Progressive Profile Level Object	$wma.profileLevelForGatedContent
ArrayList of Profile Field Objects	$wma.unsetFieldsForGatedContent

Notes:

- **profileLevelTag** – Velocity tag for Profile Level

Given a Page View object $pageView:

Integer	$pageView.destId
Integer	$pageView.objectId
String	$pageView.objectType
String	$pageView.pageType
Date	$pageView.time
Integer	$pageView.visitId
Date	$pageView.visitDate
String	$pageView.toString()

Notes:

- **objectType** – dest, content, media
- **pageType** – hub, content, profileForm

Given a Resource Portfolio object $portfolioItem:

Integer	$portfolioItem.destId
Integer	$portfolioItem.contentId

Notes:

- If isAddedByCmsUser is true, the destId will return NULL

Given a WMA Profile object $wmaProfile :

Profile Information

Integer	$wmaProfile.id
String	$wmaProfile.email
String	$wmaProfile.firstname
String	$wmaProfile.lastname
String	$wmaProfile.profileFieldTagName

Notes:

- **profileFieldTagName** – this is configured in the CMS

Update Methods

$wmaProfile.setStringValue(string *profileFieldTagName*, string *profileFieldValue*)
Can be used for text, longText, email, and multivalue fields.

$wmaProfile.setDateValue(string *profileFieldTagName*, date *profileFieldValue*)
Can be used for date and dateTime fields. Note the date saved will be the current date in the visitor timezone (the timezone returned by $wma.timeZone).

$wmaProfile.setLongValue(string *profileFieldTagName*, long *profileFieldValue*)

$wmaProfile.setBooleanValue(string *profileFieldTagName*, boolean *profileFieldValue*)

Notes:

- **profileFieldTagName** – this is configured in the CMS

Given a WMA Visitor object $wma:

Social Media

Note: these reference registered user Social Media actions such as comments, ratings, and UGC, and NOT WMA-related behavior.

For further documentation please refer to the Clickability Platform User Community at *http://community.clickability.com* > Developer Forum > Technical Docs & Webinars for the "Guide: Registered Users and Social Media".

Boolean	$wma.hasUserPerformedActionOnContent (string actionType, int contentID)
ArrayList of Comment Objects	$wma.comments
ArrayList of Ratings Objects	$wma.ratings
ArrayList of Feedback Objects	$wma.allFeedback

Notes:

- **actionType** – one of the following user action types: create_comment, edit_comment, delete_comment, create_usercontent, rate_action, ratingfeedback_action, edit_ratingfeedback, usersaved_action, all
- **contentID** – the ID of the content item

Given a WMA Visitor object $wma

Social Media Actions

Note: these reference registered user Social Media actions such as comments, ratings, and UGC, and NOT WMA-related behavior.

For further documentation please refer to the Clickability Platform User Community at *http://community.clickability.com* > Developer Forum > Technical Docs & Webinars for the "Guide: Registered Users and Social Media".

ArrayList of Action Objects	$wma.actions
ArrayList of Action Objects	$wma.getActions(int *amount*)
ArrayList of Action Objects	$wma.getActions(string *type*)
ArrayList of Action Objects	**$wma.getActions**(int *amount*, string *type*)
ArrayList of Action Objects	$wma.getActionsByContentType(int *contentTypeID*)
ArrayList of Action Objects	**$wma.getActionsByContentType**(int *contentTypeID*, int *amount*)
ArrayList of Action Objects	**$wma.getActionsByContentType**(int *contentTypeID*, string *type*)
ArrayList of Action Objects	**$wma.getActionsByContentType**(int *contentTypeID*, string *type*, int *amount*)

ArrayList of Action Objects	$wma.getActionsByContent(int *contentID*)
ArrayList of Action Objects	**$wma.getActionsByContent**(int *contentID*, int *amount*)
ArrayList of Action Objects	**$wma.getActionsByContent**(int *contentID*, string *type*)
ArrayList of Action Objects	**$wma.getActionsByContent**(int *contentID*, string *type*, int *amount*)

Notes:

- **amount** – number of actions by user
- **type** – one of the following user action types: create_comment, edit_comment, delete_comment, create_usercontent, rate_action, ratingfeedback_action, edit_ratingfeedback, usersaved_action, all
- **contentTypeID** – the ID of the content type
- **contentID** – the ID of the content item

Account Data

Accounts

ArrayList of Account Objects	$wma.accounts
Account Object	$wma.getAccountData(int accountId)
Account Object	$wma.accountData Retrieves account data for the visitor, if the visitor is associated with an account.

Notes:

- **accountId** – ID of specified account

Given an Account object $account:

String	$account.name
String	$account.description
Integer	$account.accountId
ArrayList of Account Field Objects	$account.fields
Field Object	$account.getField(string fieldName)
Boolean	$account.hasField(string fieldName)

Notes:

- **fieldName** – template tag of specified field

Given an Account Field object $accountField:

String	$accountField.dataType
String	$accountField.defaultValue
String	$accountField.fieldType
String	$accountField.label
Integer	$accountField.leadScoreContribution
Integer	$accountField.progressiveProfileLevel
String	$accountField.tag
Integer	$accountField.numberOfLines
Integer	$accountField.maxLength
Integer	$accountField.minLength
ArrayList of Select Field Options	$accountField.options
dataType Type	$accountField.value
String	$accountField.toString()

Notes:

- **dataType options** – text, longText, number, date, dateTime, email, boolean
- **fieldType options** – text, textArea, select, multiSelect
- **numberOfLines** – number of lines in textArea field type
- **maxLength** – input max length for text field type
- **minLength** – input min length for text field type

Given Select Field options $selectOption:

Integer	$selectOption.value
String	$selectOption.label

Visitor Lead Score

Website visitor behavior is tracked with the WMA lead score. Lead scores can be incremented and decremented based on the configuration of Actions and/or Content/Media Items. Once a website visitor reaches a configurable lead score trigger, a Progressive Profile Form (PPF) can be presented asking the visitor to submit more information.

Lead Score

	$wma.adjustLeadScore(int score, string description) Use this tag to adjust the score up or down. $wma.setLeadScore(int score, string description) Use this tag to replace the score with a new value.
Integer	$wma.leadScore
ArrayList of Lead Score Objects	$wma.leadScoreAdjustments() Returns all Lead Score adjustments.

Notes:

- **score** – integer amount
- **description** – description of adjustment that appears in Visitor Data reports, typically so marketers can distinguish between automated vs. manual adjustments

Given a Lead Score object $leadScoreObject:

Integer	$leadScoreObject.amount
Date	$leadScoreObject.time
String	$leadScoreObject.type
String	$leadScoreObject.description
Boolean	$leadScoreObject.isIncrement()
String	$leadScoreObject.toString()

Note:

- The Lead Score Object can be a useful developer tool for debugging; display it in development/staging environments only, in order to visually see Lead Scores as a test visitor interacts with your website.

Interests

The Lead Score and Progressive Profile help determine a website visitor's level of engagement. The Interests help determine what type of interests the visitor has. The Interests are configured in the WCM and applied to individual content and media items. Use the adjustInterestScore tag in order to track the quantity per Interest.

Interests

ArrayList of Interest Definitions	$wma.interestDefinitions
String	$wma.getInterestDefinition(string interestTag)
Integer	$wma.getInterestScore(string interestTag)
	$wma.trackInterest(string interestTag)
	$wma.setInterestScore(string interestTag, int score)
	$wma.adjustInterestScore(string interestTag, int score)

Notes:

- **interestTag** – Velocity tag name for Interest
- **score** – Integer value of adjustment amount

Given an Interest Definition object $interestDef:

String	$interestDef.name
String	$interestDef.tag
String	$interestDef.toString()

Actions

The lead score and progressive profile help determine a website visitor's level of engagement. Actions help determine what type of actions the visitor has been taking which can't otherwise be tracked via content/media items. Actions are also configured in the CMS. Lead Score adjustments are made based on the different types of activities configured.

Actions

ArrayList of ActionDefinitions	$wma.actionDefinitions
String	$wma.getActionDefinition(string actionTag)
	$wma.trackVisitorAction(string actionTag, string description)
ArrayList of VisitorAction Objects	$wma.getVisitorActionsByTagName(string actionTag)

Notes:

- **actionTag** – Velocity tag name for Action
- **description** – description for adjustment

Given an ActionDefinition object $actionDef:

String	$actionDef.name
String	$actionDef.tag
Integer	$actionDef.leadScoreAdjustment
String	$actionDef.toString()

Given a VisitorAction object $visitorAction:

String	$visitorAction.description
Date	$visitorAction.time

Content and Media

A large part of WMA includes the ability to restrict access to certain content items based on how much information we have stored about the visitor. WMA-enabled customers will see a WMA tab inside appropriate WCM content items. This tab enables users to assign an optional progressive profile level (PPL) to content items and optional lead score, as well as associated interests.

When a website visitor attempts to view a content item requiring a PPL, we will:

- Check whether the visitor has already submitted profile data for the required fields. If the visitor has all the necessary profile info, we will allow the visitor to proceed and view the content page.
- If the visitor has not submitted the necessary profile info we will:
    - Track a count for an attempted access to a protected content item.
    - Create a cookie for the visitor containing the selected content ID.
    - Redirect the visitor to the progressive profile form website section preconfigured for the domain.

Content	
ContentAttribute Object	$wma.getContentAttributes(int contentId)

Use the $wma.getContentAttributes tag to retrieve WMA values assigned to individual content items (such as lead score, interest, etc.). For example,

```
#set($contentItem = $wma.getContentAttributes(int contentId)
```

As a sample use-case, use $wma.getContentAttributes to find the interests associated with a content item, and then pass those interests to the $wma.suggestedContent tag to display content with similar interests. As another use-case, there could be a marketing strategy that involves doubling a lead score contribution for a visitor if they access a content item that has interests assigned that match the interests in the visitor's profile.

Given a ContentAttribute object $contentItem:	
ArrayList of Interest Objects	$contentItem.interestDefinitions
Integer	$contentItem.leadScoreContribution
Progressive Profile Level Object	$contentItem.profileLevel
Integer	$contentItem.contentId
ArrayList of Campaign Objects	$contentItem.campaignIds
Given a Campaign object $campaign:	
String	$campaign.Id

Media	
MediaAttribute Object	$wma.getMediaAttributes(int mediaId)

Use the $wma.getMediaAttributes tag to retrieve WMA values assigned to individual media items (such as lead score, interest, etc.). Refer to the $wma.getContentAttributes examples above for context.

Given a MediaAttribute object $mediaItem:

ArrayList of Interest Objects	$mediaItem.interestDefinitions
Integer	$mediaItem.leadScoreContribution
Interest	$mediaItem.mediaId
String	$wma.getMediaTrackingUrl(MediaObject media) Use this tag to automatically track the Lead Score Contribution and Interest Assignments as an alternative to the tags above. Example: #set($sampleMediaId = 12345) #set($noTrackURL = $cms.media($sampleMediaId).url) #set($trackURL = $wma.getMediaTrackingUrl($cms.media($sampleMediaId)))

Suggested Content Based on Interests

The following tags will return content items which may be suggested to the visitor as determined from the visitor's current interests. It is intended (but not required) that the priority of interests be based on the visitor's interest scores. The Suggested Content algorithm first tries to match content that has all of the passed interests, if it finds enough content to suffice the request count, it will return those items; if not, it removes one interest from the tail end of the interest list and tries to match again.

ArrayList of Content Items	$wma.suggestedContent For this tag, interests associated with the visitor are used to surface suggested content.
ArrayList of Content Items	$wma.getSuggestedContent(integer numItems)
ArrayList of Content Items	$wma.getSuggestedContent(arrayList interestTags[, integer numItems])
ArrayList of Content Items	$wma.getSuggestedContent(arrayList interestTags, boolean excludeViewedContent[, integer numItems])

Notes:

- **numItems** – the number of items to display; the default is 10 if numItems is not specified
- **interestTags** – an ArrayList of Interest tags
- **excludeViewedContent** – by default, content items the visitor has already viewed are automatically excluded

Search

The following tags can be used to track a site visitor's search behavior.

$wma.trackSearch(string searchTerms, boolean resultReturned)

Integer	$wma.searchCount

Notes:

- **searchTerms** – this will track the keywords a visitor has searched for
- **resultReturned** – if this is true, the first result a visitor clicks on will be tracked

Example:

```
#set($searchKeywords = $req.param("keywords"))
#if($mySearch.execute.size() == 0)##
	#set($foo = $wma.trackSearch($searchKeywords, false))
#else
	#set($foo = $wma.trackSearch($searchKeywords, true))
#end
```

Campaign Tracking

WMA tracks campaign touchpoints for website visitors based on Campaigns and Campaign Sources that are configured in the WCM interface. The data is made available in various Campaign reports and can be exported along with visitor data.

The tags below are intended for use-cases when you need to manually track campaign activity.

$wma.trackCampaign(string campaignId, string description)

ArrayList of CampaignDefinitions	$wma.campaignDefinitions

Notes:

- **campaignId** – ID of campaign
- **description** – description of campaign

Given a CampaignDefinition object $campaignDefinition:

String	$campaignDefinition.name
Integer	$campaignDefinition.campaignId
Date	$campaignDefinition.startDate
Date	$campaignDefinition.endDate
String	$campaignDefinition.paramKey The URL parameter name

Publisher Tools

User location data

This provides access to the website visitor geo-location data.

User Location - $userLocation

Boolean	$userLocation.isAvailable()
String	$userLocation.ip
String	$userLocation.areaCode
String	$userLocation.city
String	$userLocation.country
String	$userLocation.isp
String	$userLocation.latitude
String	$userLocation.longitude
String	$userLocation.metropolitanCode
String	$userLocation.organization
String	$userLocation.postalCode
String	$userLocation.region
String	$userLocation.timeZone

Notes:

isAvailable() returns true if geolocation data can be obtained, false otherwise timeZone returns the time zone in long form (ie; "America/Los_Angeles") and can be used in $util context.

HTTP Request and Response

The $req utility enables access to HTTP request details such as parameters, request headers, cookies, etc. The $req context is available to all customers, however these additional $req context methods were developed specifically for the WMA non-cached environment.

User Request - $req

String	**$req.param**(string *name*[, string *defaultValue*])
Integer	**$req.intParam**(string *name*[, string *defaultValue*])
Long	**$req.longParam**(string *name*[, string *defaultValue*])
Float	**$req.floatParam**(string *name*[, string *defaultValue*])
Double	**$req.doubleParam**(string *name*[, string *defaultValue*])
Date	**$req.dateParam**(string *name,* string *dateFormatPattern,* string *timeZone*) Parses the specified parameter value using the supplied date format pattern and timezone.
Boolean	**$req.booleanParam**(string *name*[, string *defaultValue*]) Returns true if the parameter value is "y", "true", or "1" (case insensitive), false otherwise.
String	$req.paramValues(string *name*)
Collection	$req.params
String	$req.referrer or $req.referer Returns the “Referer” header value.
String	$req.ip Returns the client IP address as a string.
String	$req.userAgent Returns the “User-Agent” header value.
String	**$req.header**(string *header*) Returns the named request header value, or null if the header was not set in the request.
String	$req.acceptLanguage Returns the “Accept-Language” header value.

Notes:

- **defaultValue** – this can be specified in case the parameter has no value; it is not required
- timeZone returns the time zone in long form (ie; "America/Los_Angeles")

The $userRequest context will include all of the above $req context methods, with the addition of the following:

User Request - $userRequest

String	$userRequest.getCookie(string *cookieName*)
String	$userRequest.referrerDomain Returns the domain/host portion of a URL set in the "Referer" header. Will return null if the value is not a URL or is not set.
Object	$userRequest.referrerSearchEngine

Given a User Request object $userRequestObject:

String	$userRequestObject.name
String	$userRequestObject.keywords

Notes:

- **cookieName** – the string name of the cookie to retrieve

Example:

```
#if($userRequest.referrerSearchEngine)
       Search engine: $userRequest.referrerSearchEngine.name<br>
       Keywords: $userRequest.referrerSearchEngine.keywords<br>
#end
```

The $userResponse will set HTTP headers on the response, including cookies. All use of these methods must be done early in the page generation. All methods return a boolean value which will be true if the header or cookie could be set, false if not.

User Response - $userResponse

Boolean	$userResponse.isCommitted() Returns true if the response was already committed meaning that headers and cookies can no longer be set.
Boolean	**$userResponse.setHeader**(string *name*, string *value*)
Boolean	**$userResponse.setIntHeader**(string *name*, integer *value*)
Boolean	**$userResponse.setDateHeader**(string *name*, java.util.Date *value*)
Boolean	**$userResponse.setCookie**(string *name*, string *value*, int *maxAgeInSeconds*) Cookie domain will be set with "." appended to the front of the current publishing domain.
Boolean	$userResponse.removeCookie(string *name*)

User Agent Information

The $userAgent utility allows access to the site visitor's user agent information.

User Agent - $userAgent Various Methods	
String	$userAgent.OSDetail
Int	$userAgent.geckoVersion Returns the gecko major version *10 as an integer. So Gecko rv:1.7.8 would return 17, and Gecko rv:1.8.0.1 would return 18, etc.
String	$userAgent.fullGeckoVersion Returns the full gecko version as a string.
String	$userAgent.browserDetail
String	$userAgent.userAgent

User Agent - $userAgent Bot Detection	
Boolean	$userAgent.isBot() Returns true if the request is from one of the search engine spiders listed below.
Boolean	**$userAgent.isGooglebot**()*includes Google Image bot
Boolean	$userAgent.isMSNBot() *use for Bing*
Boolean	$userAgent.isWebCrawler()
Boolean	$userAgent.isInktomi()
Boolean	**$userAgent.isAsk**() *Ask/Ask Jeeves/Teoma
Boolean	$userAgent.isBaidu() *Chinese*
Boolean	$userAgent.isCuil()
Boolean	$userAgent.isSogou() *Chinese*
Boolean	$userAgent.isSohu() *Chinese*
Boolean	$userAgent.isYandex() *Russian*

User Agent - $userAgent OS Detection

Boolean	**$userAgent.isWindows**() *all non-mobile versions
Boolean	$userAgent.isWin()
Boolean	$userAgent.isWin7()
Boolean	$userAgent.isWinVista()
Boolean	$userAgent.isWinXP()
Boolean	$userAgent.isWin2003()
Boolean	$userAgent.isWin2000()
Boolean	$userAgent.isWinNT()
Boolean	$userAgent.isWinME()
Boolean	$userAgent.isWin98()
Boolean	$userAgent.isWin95()
Boolean	$userAgent.isMac()
Boolean	$userAgent.isMacOSX()
Boolean	$userAgent.isMacPPC() *includes OS 9*
Boolean	$userAgent.isLinux()
Boolean	$userAgent.isBSD()
Boolean	$userAgent.isSunOS()
Boolean	**$userAgent.isMobile**() *only detects common devices
Boolean	**$userAgent.isiPhone**()*includes iPod Touch*
Boolean	$userAgent.isPalmOS()
Boolean	$userAgent.isBlackBerry()
Boolean	$userAgent.isPocketPC()
Boolean	$userAgent.isAndroid()

User Agent - $userAgent Browser Detection

Boolean	**$userAgent.isIE**() *all non-mobile IE versions
Boolean	$userAgent.isIE8()
Boolean	$userAgent.isIE7()
Boolean	$userAgent.isIE6()
Boolean	$userAgent.isGecko()
Boolean	$userAgent.isNetscape()
Boolean	$userAgent.isChrome()
Boolean	$userAgent.isIEMobile()
Boolean	**$userAgent.isOperaMini**()* all Mobile Opera versions
Boolean	$userAgent.isBlazer()
Boolean	$userAgent.isBlackBerryBrowser()
Boolean	$userAgent.isMobileSafari()
Boolean	$userAgent.isSafariMobile()
Boolean	$userAgent.isAndroidBrowser()
Boolean	$userAgent.isSafari() *non-mobile*
Boolean	**$userAgent.isFirefox**() *all Firefox versions*
Boolean	$userAgent.isFirefox2()
Boolean	$userAgent.isFirefox3()
Boolean	$userAgent.isFirefox30()
Boolean	$userAgent.isFirefox35()
Boolean	$userAgent.isOpera()

WMA Code Samples

Progressive Profile Level:
First, setup some variables:

```
#set($visitorScore = $wma.leadScore)
#set($visitorLevel = $wma.profileLevel.level)
#set($visitorLevelName = $wma.profileLevel.name)

#if(!$visitorLevelName)
	#set($visitorLevelName = "Anonymous")
#end
#if(!$visitorLevel)
	#set($visitorLevel = 0)
#end

#foreach($level in $wma.getProfileLevels())
	#set($profileTrigger = $level.scoreTrigger)
	#set($profileLevel = $math.newInt($level.level))
	#set($profileLevelTag = $level.tag)
	#set($nextProfileLevel = $math.add($visitorLevel, 1))

	## Figure out the next level
	#foreach($nextLevel in $wma.profileLevels)
		#if($nextProfileLevel == $math.newInt($nextLevel.level))
			#set($nextProfileLevelTag = $nextLevel.tag)
			#set($nextProfileLevelName = $nextLevel.name)
		#end
	#end
```

Next, make sure the form only displays when necessary (content gating or lead score trigger):

```
	#if($level.level > $visitorLevel)
	#if((($visitorScore >= $profileTrigger) || $cms.path == "/en/update")
&& (!$formDisplay || $formDisplay != "yes"))
		##Get the next set of PPL fields to display for the visitor

	#if($wma.getUnsetFieldsForProfileLevel($nextProfileLevelTag).size()
!= 0)
		## The following is javascript used to post multi-select fields
		<script>
		function doAction(){
		  var selectedVals = '';
		  var selectCtrl = document.forms['profileForm'].multiSelect;
		 for(var index=0; index < selectCtrl.options.length ; index++){
			if(selectCtrl.options[index].selected){
			selectedVals = selectedVals +
selectCtrl.options[index].value + ',';
		  }
		}
		if(selectedVals.length > 0 &&
selectedVals.charAt(selectedVals.length-1) == ',')
			selectedVals =
selectedVals.substr(0,selectedVals.length-1);
			document.forms['profileForm'].multiAnswer.value =
selectedVals;
		}
		</script>
		<table style="background-color:lightgreen;">
		## setup PPL
```

```
            <form name="profileForm" method="post" onsubmit="return
doAction();" action="/s">
            <input type="hidden" name="action" value="submitProfile" />
            <input type="hidden" name="rurl" value="$req.fullURL" />
            <input type="hidden" name="multiAnswer" value=""/>
```

Add in some error handling:

```
                  #if($subscription.error)
                        <div class="subscriptionError">
                        DefaultMsg:<font
color="red">$!subscription.errorMessage</font>
                        </div>
                        <div class="subscriptionError">
                        ErrorCode:<font
color="red">$!subscription.errorCode</font>
                        </div>
                        <div class="subscriptionError">
                        ErrorDetails:<font
color="red">$!subscription.errorDetails</font>
                        </div><br>
                  #end
                  <input type="hidden" name="errorDestId"
value="$cms.websiteSection.id" />
                  #if($cms.content)
                        <input type="hidden" name="errorContentId"
value="$cms.content" />
                  #end
```

Display unset fields being sure to display the correct input for each data type:

```
#foreach($field in $wma.getUnsetFieldsForProfileLevel($nextProfileLevelTag))
      #if($string.indexOf($field.dataType, "date") != -1)
            <tr><td>$field.label : </td><td><input type="date"
name="$field.tag"/></td></tr>
      #elseif($string.indexOf($field.dataType, "number") != -1)
            <tr><td>$field.label : </td><td><input type="integer"
name="$field.tag"/></td></tr>
      #elseif($field.fieldType == "select")
            <tr><td>$field.label : </td><td><select name="$field.tag"/>
            #foreach($option in $field.options)
                  <option value="$option.value">$option.label</option>
            #end
            </select>
            </td></tr>
      #elseif($field.fieldType == "multiSelect")
            <tr><td>$field.label : </td><td><select multiple
name="multiSelect" />
            #foreach($option in $field.options)
                  <option value="$option.value">$option.label</option>
            #end
            </select>
            </td></tr>
      #else
            <tr><td>$field.label : </td><td><input type="text"
name="$field.tag"/></td></tr>
      #end
#end
<tr><td><input type="submit" value="Submit"/></td></tr>
</form>
#set($formDisplay = "yes")
```

```
#end </table><br>
#end
#end
```

Refer to the User Community at **http://community.clickability.com** **> Developer Forum > Velocity Code Library** for more WMA code samples.

Miscellaneous Tags

HTTP Client Service

#set($myNewClient=$httpClient.newHttpClient())

This creates a new HTTP client object that can be edited.

String	$myNewClient.setUrl(string url)
String	$myNewClient.setRequestType(string type)
String	$myNewClient.setRequestEntity(string content, string contentType) To be used with PUT and POST to allow sending to allow sending XML, JSON and other non application/x-www-form-urlencoded encoded payloads. **PUT** For PUT this is the only way to set the request body. **POST** a) For POST if setRequestEntity() is NOT called the payload will be of content-type application/x-www-form-urlencoded and will be made up of all the the name value pairs added by calling $httpClient.addRequestParameter(). b) For POST if setRequestEntity() IS called the payload will be the value specified by the content argument and the content-type will be that specified by the content-type argument. Any calls to $httpClient.addRequestParameter() will be ignored.
String	$myNewClient.addRequestHeader(string name, string value) Headers are assigned to the HTTP request as name/value pairs. A user can specify as many headers as necessary.
Parameter	$myNewClient.addRequestParameter(string name, string value) Parameters are configured as name/value pairs. A user can specify as many headers as necessary and the request will be automatically generated in the appropriate format depending on the specified Type for GET, PUT, DELETE and HEAD calls. Only POST call will not have these appended to the URL as a query string.
String	$myNewClient.addResponseHeader(string httpHeader) This tag will inspect the HTTP response for the specified header and include it in an array list or collection of headers included in the response.
Integer	$myNewClient.setTimeout(int time)
Integer	$myNewClient.setTTL(int ttlMinutes)
String	$myNewClient.execute() This tag executes the request.
String	$results.responseString
ArrayList	$results.responseHeader
String	$results.responseCode The HTTP Response Code returned from the server which can be useful to validate that your request completed as expected.

Usage:

- **url** – The request URL for the client.
- **type** – The type of request to be made – GET, POST, PUT, DELETE or HEAD.
- **httpHeader** – The specified HTTP header.
- **time** – The time, in seconds, to timeout the request. The default timeout is 5 seconds.
- **ttlMinutes** – The production cache TTL in minutes on the HTTP response if remote URL caching is enabled.

Note: HTTP responses to POST, PUT and DELETE requests are never cached and any TTLs set ignored. Calling $httpClient.setTTL(0) or not calling $httpClient.setTTL() prevents a response from a GET or HEAD request from being cached. Note TTL is not honored in development or stage environments.

Accessing the Results

Use `#set($results = $myNewClient.execute()` to execute the request.

Once you get the results, there are two methods available:

```
$results.responseString
```

This returns the results as a string.

```
$results.responseHeader
```

This returns a collection of header names (.name) and values (.value) included in the HTTP response. If the headers were specified, only those will appear in the ArrayList, otherwise all will be available. Loop through the ArrayList to get the name and value pairs, such as:

```
#foreach($header in $results.responseHeader)
$header.name - $header.value<br/>
#end
```

Code Sample:

```
#set($myNewClient= $httpClient.newHttpClient())
#set($foo = $myNewClient.setUrl("http://www.google.com/search"))
#set($foo = $myNewClient.setRequestType("GET"))
#set($foo =
$myNewClient.addRequestHeader("requestHeader1","requestHeader1Value"))
#set($foo =
$myNewClient.addRequestHeader("requestHeader2","requestHeader2Value"))
#set($foo = $myNewClient.addRequestParameter("hl","en"))
#set($foo =
$myNewClient.addRequestParameter("q","post+url+with+parameters+to+google"))
#set($foo = $myNewClient.addRequestParameter("aq","f"))
#set($foo = $myNewClient.addResponseHeader("Cache-Control"))
#set($foo = $myNewClient.addResponseHeader("Content-Type"))
#set($foo = $myNewClient.setTimeout(12))
#set($results = $myNewClient.execute())

$results.responseString - <br>
#foreach($r in $results.responseHeader)
    $r.name - $r.value<br>
#end
```

Access to Outside URLs

String	$util.getURL(string url) IMPORTANT: In most cases, we recommend using $httpClient.newHttpClient() rather than $util.getURL. For very lightweight use-cases involving a one-line, one argument usage, $util.getURL may be appropriate. This tag is used to statically include the content from another web page into the page being published. It will close the connection if not answered within 10 seconds.
String	$util.getURLWithTimeout(string url, int timeout)
String	$util.getURLWithTimeout(string url, int timeout, long cacheTtl)
Parameter	$util.newParam(string name, string value)
String	$util.postURL(string url, ArrayList params) IMPORTANT: In most cases, we recommend using $httpClient.newHttpClient() rather than $util.postURL. For very lightweight use-cases involving a one-line, one argument usage, $util.postURL may be appropriate. This tag is used to make an http POST request to another site, passing an ArrayList of Parameters (name value pairs). The response from that URL will then be included into the page being published. This tag will timeout after 10 seconds.

Usage:

- **url** – The URL of the web page to included.
- **timeout** – The length of the timeout from 5 (min) to 20 (max) seconds.
- **cacheTtl** – The specified cache ttl.

Remote Server Manager

External HTTP services accessed via the various Publisher "getUrl" tags have historically been one of the most problematic parts of page generation. Because the availability and stability of such services is completely out of our control, many problems can be caused when such services become unresponsive. The Remote Service Manager provides a way to group all calls to a remote service within a manager that can control whether such remote calls should be blocked or should be allowed to proceed. The feature also provides a way for the service manager to auto-detect that the service has gone down, as well as automated self-recovery. The rules upon which such automated actions are taken are fully configurable.

A remote service is defined uniquely in the following way:

- The customer.
- The published domain.
- The domain name of the remote service.
- The root path of the remote service, to which included URLs can be matched.

This definition allows for a remote service to be configured separately for each site on which that service is used.

Once a remote service has been defined, it is managed though a "Remote Service Manager".

Defining a Remote Service Manager

Currently a remote service manager can only be defined via a special template tag. The logical next step for this feature is to persist such definitions and provide some sort of UI for definition and configuration.

```
#set($feedServiceManager =
$util.defineRemoteService("www.feedprovider.com","/feeds"))
```

This tag is intended to be placed in a template which would automatically be included in every generated page for a site. When this method is called, any previously defined service for the same domain and path will simply be returned.

The returned object can be used to set various configuration parameters as described below.
To un-define a service, the returned service manager can be disabled as described below. Although the service is still defined, it will effectively behave as if were never defined in the first place.

Automated Failure and Recovery

When various configurable conditions are found to occur for one or more remote requests, the service manager will transition the service into a "Down" status. When the service is down, any subsequent calls to the service will be immediately canceled, meaning that the published page will either be returned an empty response, or a previously cached version of the page depending on which client context is making the call.

Currently failure can be configured to occur for various types of exceptions, or for 500 level HTTP response codes. It is also possible to configure a threshold for failure, including how many failures must occur within a configurable period of time.

Once a service is taken down, the service manager will enter into an automatic recovery mode. Periodically, a single client request will be allowed to proceed. If that request succeeds, meaning that the response did not meet any of the failure conditions, the service will be brought back up.

Configuring a Remote Service Manager

The object returned by the defineRemoteService method can be used to dynamically configure how the remote service manager behaves. The following methods can be called on the returned service manager object. Each of the setter methods simply returns back a reference to the same service manager object.

Configuration Method	Description	Default Value
setDisabled()	Disables the service manager. Any matching URLs will be executed as if the service were never defined.	False
setEnabled()	Re-enables the service manager so that matching URLs will be managed by the service.	True
setDown()	Flags the service as down. All remote connections to URLs matching this service will be blocked. When this is used, the automatic recovery mechanisms will not be active.	False
setUp()	Flags the service as up. Remote connections to URLs managed by the service will be allowed to proceed.	True
setFailOnConnectionTimeout(boolean *failOnTimeout*)	Sets whether a connection timeout should be used as a failure condition. A connection timeout occurs when a connection to the remote service could not be established in the configured period (internal Clickability configuration).	True

setFailOnSocketTimeout(boolean *failOnTimeout*)	Sets whether a socket timeout should be used as a failure condition. A socket timeout occurs when a connection to the remote service can be made, but the service does not complete sending its response data within the configured amount of time (internal Clickability configuration).	True
setFailOnConnectionManagerTimeout(boolean *failOnTimeout*)	Sets whether a connection manager timeout should be used as a failure condition. This occurs when the HTTP connection manager does not have a free connection available in its pool. This is not recommended to be used as a failure condition because the http connection pool is shared by many services, so it is really not a good indicator of whether this service is down.	False
setFailOnIoException(boolean *failOnIoException*)	Sets whether a general IO exception should be used as a failure condition.	False
setFailOnHttpException(boolean *failOnHttpException*)	Sets whether an HTTP exception should be used as a failure condition.	False
setFailOn500Response(boolean *failOn500*)	Sets whether a 500 level HTTP response code (Server Error) should be used as a failure condition.	False
setFailOn400Response(boolean *failOn400*)	Sets whether a 400 level HTTP response code (such as 404, Not Found) should be used as a failure condition. This should not be used in production, but may be very useful for testing, as it is very simple to simulate a failure by requesting a non-existent URL which would return a 404 error.	False
setRetrySeconds(int *retrySeconds*)	The number of seconds after a service is taken down by the automatic failure detection, after which a recovery attempt will be made.	60
setFailureThresholdCount(int *failureThresholdCount*)	The number of request failures which must occur before the service is taken down.	1
setFailureThresholdSeconds(int *failureThresholdSeconds*)	When the threshold count is > 1, the number of seconds within which the configured number of failures must occur before the service is taken down.	5
setMaxConnections(int *maxConnections*)	The maximum number of threads which will be allowed to concurrently make requests to the remote service. Once exceeded, additional requests will either be canceled immediately or with a configured timeout.	20
setConnectionTimeoutSeconds(int *timeoutInSeconds*)	The amount of time in seconds which clients should wait for a free connection when all of the available connections are in use. If a value of 0 is specified, the request will be canceled immediately.	0
setConnectionTimeoutMillis(long *timeoutInMilliseconds*)	The amount of time in seconds which clients should wait for a free connection when all of the available connections are in use. If a value of 0 is specified, the request will be canceled immediately.	0

The following methods can be used to read the current configuration:

```
isEnabled()
isDisabled()
isUp()
isDown()
isFailOnConnectionTimeout()
isFailOnConnectionManagerTimeout()
isFailOnSocketTimeout()
isFailOnIoException()
isFailOnHttpException()
isFailOn500Response()
isFailOn400Response()
retrySeconds
failureThresholdCount
failureThresholdSeconds
maxConnections
connectionTimeoutSeconds
connectionTimeoutMillis
```

Service Control Template

```
## Define the service; this sample is for a blog service. It is safe for
this call to be made multiple times (such as every time a page is
generated). A previously defined service will be returned if one exists.
#set($blogsService = $util.defineRemoteService("blogs.domain.com","blogs"))

## Set various configuration options
#set($junk = $blogsService.setRetrySeconds(60))
#set($junk = $blogsService.setMaxConnections(20))
#set($junk = $blogsService.setFailOnConnectionTimeout(true))
#set($junk = $blogsService.setFailureThresholdCount(3))
#set($junk = $blogsService.setFailureThresholdSeconds(10))

## Uncomment this line to disable the service manager (feed requests will no
longer be managed by the service)
## #set($junk = $blogsService.setDisabled())

## Uncomment this to re-enable the service manager (if it was previously
disabled).
## #set($junk = $blogsService.setEnabled())

## Uncomment this line to force the service to go down. Auto-recovery
process will not be performed.
## #set($junk = $blogsService.setDown())

## Uncomment this line to force the service to go up.
## #set($junk = $blogsService.setUp())

<!-- Current Blogs Service Status: -->
<!-- Remote Service: $blogsService -->
<!-- Status: #if($blogsService.isDisabled()) Disabled
#elseif($blogsService.isDown()) Down #else Up #end <br><br> -->
```

Once this template is included in every page generated, it can be used to control or reconfigure the service manager. For instance, if the service were to fail, but the auto-failure detection was not working, the template could be edited so that the `setDown()` method call was uncommented. This would effectively block all remote calls to URLs matching the service. When the service is ready to resume taking connections, the template could be edited again to uncomment the `setUp()` call.

Testing Remote Service Managers

This feature may be very difficult to set up for testing. One difficulty will be simulating connection timeouts or other "real" failure conditions. As a workaround for this, refer to the 400 level failure condition 'setFailOn400Response'. If this is enabled, you can simply request a page which would normally return a 404. If the service is configured to fail for 400 level errors, then requesting such a page will take it down.

Redirect via Velocity

String	$cms.redirect(string url[, boolean isPermanent]) When called during a page generation and the response is not already committed, template generation will halt and the user will be redirected to the specified URL. Note these tags will only work if the page is not already committed, or at the beginning of a page generation, in the same way as the $cms.setHeader methods work. If the reponse is already committed, the template debug logs will indicate the redirection failed.

Usage:

- **url** – The URL of the web page to be redirected to.
- **isPermanent** – Set to 'false' for a temporary 302 redirect. If left blank, the redirect will default to a permanent 301 redirect.

Visitor Location Data Based on IP Address

A series of tags are available that provide access to the website visitor geo-location data. Note these tags are only appropriate for non-cached pages, (such as subscription pages at /s) and it is necessary for Client Services to activate the "Geo Location Service" for your account prior to use.

For the list of tags, go to the `$userLocation` section of this manual within the WMA Chapter.

Translation Management using Content Hierarchies

The following methods have been tailored for template developers to retrieve a list of all translations for a given content item and determine if a translation exists, as well as to determine the "closest translation" based on the Content Hierarchy Definition.

Given a ContentHierarchyElement object $element:

A ContentHierarchyElement object represents a content item and metadata associated with the content item via the Content Hierarchy instance it belongs to.

String	$element.nodeName The name of the node in the Content Hierarchy Definition that this content item item is associated with. In the translation context this should be the name of the translated language.
Integer	$element.contentId
String	The ID of the content item that this object represents.

Usage:	$element.translationStateName One of the following states: Original Document Not Requested Requested In Progress Complete

The template tags below mention that in certain cases they return or do something if a translation "**exists**" for a content item. Here is what that means:

- The translation management feature is built on top of the core Content Hierarchy functionality. Content Hierarchy considers an associated content item to exist if a content item is associated with that node in the underlying Content Hierarchy instance.
- A translation is considered "to exist" from the moment it is requested and a content item is created to represent that item in the Content Hierarchy instance. The item can be at any state in the translation management workflow (e.g. 'Requested', 'Not Requested', 'In Progress' or 'Complete').

The following best practices will help ensure content only appears on a site once the translation process is complete and approved:

- A content item status should be set to 'Ready' (or the final workflow state) only after an item reaches the translation state of "Complete".
- Template developers should always filter any content items returned to discard any that are not in the "Ready" state.

All translation management functionality can be accessed via the $translationMgmt variable:

ArrayList of ContentHierarchyElement Objects	$translationMgmt.translations(int contentID) Returns a list of objects that represent a translation of the specified content item for languages where a translation exists. (See note above regarding "exists".) If no translations exist an empty list will be returned (i.e. a list with the size of 0).
ArrayList of ContentHierarchyElement Objects	$translationMgmt.translations(int contentID) Returns a list of objects that represent a translation of the specified content item for languages where a translation exists. (See note above regarding "exists".) If no translations exist an empty list will be returned (i.e. a list with the size of 0).
Content Hierarchy Element	$translationMgmt.translation(string language-name, int contentID) Returns the Content Hierarchy Element that represents the translation of the content item specified in the language specified. If no translation exists for this language, null is returned. Use $translationMgmt.closestTranslation() to find the closest translation and recall this method.

ArrayList of Strings	$translationMgmt.translatedLanguages(int contentID) Returns a list of Strings where each String is the language name where a translation exists for the content. (See note above regarding “exists”.)
Boolean	$translationMgmt.doesTranslationExist(string language-name, int contentID) Returns true if a translation in the specified language exists for the specified content item; otherwise returns false. (See note above regarding “exists”.)
String	$translationMgmt.closestTranslation(string language-name, int contentID) Based on the hierarchical language relationship, this method finds the “closest” translation to the specified language for the specified item, and returns a String where the String is the language name as defined in the Content Hierarchy Definition.

Usage:

- **language-name** – the name of the language as defined in the associated Content Hierarchy definition. Language names are case-sensitive; if the language name is misspelled, the name ‘null’ will be returned.

Template Debugging Messages

Message	Log Level	Purpose
An invalid node name was supplied	ERROR	When calling one of doesTranslationExist(), closestTranslation(), or translation() an invalid language name was specified, i.e. there is no node with that name in the associated Content Hierarchy Definition.
Cannot find associated content hierarchy for content with ID	ERROR	The content item supplied was invalid and therefore an empty set of results was returned. If the content item specified is not associated with a Content Hierarchy Instance then this message is logged.

Code Samples

```
#set($contentId = $req.intParam("contentId"))
#set($translations = $translationMgmt.translations($contentId))

<h3>Translations for $contentId</h3>
  #foreach($translation in $translations)
    Translation: $translation.nodeName $translation.contentId<br/>
  #end

<h3>French Translation for $contentId</h3>
#if ($translationMgmt.doesTranslationExist("French", $contentId))
#set($frenchTranslation = $translationMgmt.translation("French",
$contentId))
French Translation Exists: $frenchTranslation.nodeName
$frenchTranslation.contentId<br/>
#else
No French Translation Exists
#end

<h3>Translated Languages</h3>
#set($translatedLanguages =
$translationMgmt.translatedLanguages($contentId))
  #foreach($language in $translatedLanguages)
    Language: $language<br/>
  #end

<h3>Closest Translation</h3>
#set($closestLanguage = $translationMgmt.closestTranslation("Catalan",
$contentId))
$closestLanguage
```

Non-Translation Content Hierarchies

The following methods can be used alternatively with the $translationManagement context (above) for content hierarchy use-cases that do not have translation flags enabled.

Given a ContentHierarchy object $contentHierarchy:	
HashMap	$contentHierarchy.getAssociatedContent(int contentId) Returns a Map where the keys are Strings that represent Node Name in the Content Hierarchy Definition and values are instances of ContentHierarchyMapValue objects that represent all content associated with the supplied content item via a Content Hierarchy Instance and meta data about that relationship. This method returns null if there is not Content Hierarchy Instance to which this specified contentId belongs
ArrayList	$contentHierarchy.getNodesNamesWithAssociatedContent(int contentId) Returns a list of node names that have content that's related to the specified content item.
Boolean	$contentHierarchy.doesAssociatedContentExistAtNode(string nodeName, int contentId) Returns true if a content item has been associated with the given node in the content hierarchy that the content item specified by contentId belongs to.
String	$contentHierarchy.getClosestNodeNameWithAssociatedContent(string nodeName, int contentId) Returns the name of the closet node with associated content in the content hierarchy instance that the content item specified by the contentId belongs to.

Code Sample

Loop through keyset ($contentHierarchy.getAssociatedContent($contentId).keySet()):

```
#foreach($nodeName in
$contentHierarchy.getAssociatedContent($contentId).keySet())
       #set($node =
$contentHierarchy.getAssociatedContent($contentId).get($nodeName)) Node:
$node.nodeName - $node.contentId
#end
```

“In Context” Editing Tags

The “Quick Links” and “Website View” features enable WCM users to link directly into content items for contextual editing:

- Quick Links provides links FROM individual content items on a webpage TO the selected items in the WCM interface
- Provides a special view of the website from within the WCM interface that enables direct editing of text/HTML fields (including boolean and select list fields)

These contextual editing features allow WCM users to quickly update items while reviewing them on the website. The Quick Links mode is accessed via a parameter:

Automatic	?inContext=auto	When a published page is rendered, the publisher attempts to identify all places where text and HTML fields from content items are used and then automatically insert: a) an edit link to the supplied content to the left of the content item via a “pencil” icon b) enclose the text in a tag to allow effects such as underlining when rollover over the edit icon
Note: the “manual” Quick Links mode has been deprecated and is no longer supported. As of the Walrus Release in April 2011, ?inContext=y will act the same as ?inContext=auto.		
Template tag based (Manual)	?inContext=y	When a published page is rendered in this mode, controls are only added that are generated from template tags. The publisher will automatically add only the following: 1) links to JavaScript and CSS in the Document Head that is needed to generate the controls 2) a set of tags that are inserted at the beginning of the Document Body that define the markup for the metadata rollovers

The Website View mode is accessed via the WCM > Publish Tab > Website View.

The following set of template tags creates controls when generating a page via the Quick Links or Website View modes. The purpose of these tags is to allow template developers to have control over where they wish the contextual editing controls to appear.

Boolean	$inContextEdit.isInContextEditMode() `Returns true if you are in Quick Links or Website View modes.`
Boolean	$inContextEdit.isQuickLinks() Returns true if you are in Quick Links mode.
Boolean	$inContextEdit.isWebsiteView() Returns true if you are in Website View mode.
	$inContextEdit.createContentQuickLink(string fieldTemplateTag, int contentID[, boolean useInAutomaticMode]) Creates a Quick Links control that provides a link to the supplied content item and provides a box of metadata about the content item. **Note:** the original fieldLabel parameter is no longer required nor supported.
	$inContextEdit.enableContentTypesOnly(string csv) Invoke this tag in the first template called on a published page by supplying a comma separated list of content type IDs, to restrict the Quick Links controls to the set of content types in the list, e.g. $inContextEdit.enableContentTypesOnly("1,2,3").
	$inContextEdit.enableFieldsOnly(string csv)

Invoke this tag in the first template called on a published page by supplying a comma separated list of field template tags, to restrict the Quick Links controls to the set of fields with a matching template tag across all types, e.g. $inContextEdit.enableFieldsOnly("title,html").

$inContextEdit.processUrl(string path)

a) In Quick Links mode this returns the URL with inContextEdit=y appended as a query string parameter

b) In Website View mode this returns a piece of JavaScript of the form: CLICKINCTXEDIT.navigateToAnotherPage(<rewritten and encoded URL>); this is necessary since Website View uses JavaScript to process links inorder to alert users if they are navigating offsite or from a page with unsaved changes

Safe Mode

The following tags support disabling Quick Links markup injection, for example when using the output of content in JavaScript or CSS or in HTML tag attributes.

$inContextEdit.beginSafeSection()

Marks the beginning of a “safe” section where no markup for Quick Links or Website View will be injected.

$inContextEdit.endSafeSection()

Marks the end of a “safe” section where no markup for Quick Links or Website View will be injected

$inContextEdit.safe(object o)

Disable Quick Links or Website View markup injection for a particular variable

Usage:

- **fieldTemplateTag** – Template tag of the content type field the text was derived from
- **contentID** – Content ID# of the content item where the text came from
- **useInAutomaticMode** – Setting this to true enables this tag in automatic mode, e.g. ?inContext=auto
- **path** – the URL to be rewritten and encoded for use with Website View, e.g. “/news” or “http://www.yahoo.com”
- **o** – Particular variable representing a content type field, e.g. $inContextEdit.safe($title)

Code Samples

```
#if($inContextEdit.isQuickLinks() || !$inContextEdit.isInContextEditMode())
<span
onclick="window.location.href='$!inContextEdit.processUrl("/news")'">(Quick
Links) Click Here to Go the News Section via an onclick Event</span>
<br/><br/>
<span
onclick="window.location.href='$!inContextEdit.processUrl("http://www.yahoo.
com")'">(Quick Links) Click Here to Go the www.yahoo.com via an onclick
Event</span>
#elseif($inContextEdit.isWebsiteView())
<span onclick="$!inContextEdit.processUrl("/news")">(Website View) Click
Here to Go the News Section via an onclick Event</span>
<br/><br/>
<span onclick="$!inContextEdit.processUrl("http://www.yahoo.com")">(Website
View) Click Here to Go the www.yahoo.com via an onclick Event</span>
#end
```

Publisher Encryption Utilities

The acceptable algorithm, mode, and padding values are specified in the JCE documentation here:
http://download.oracle.com/javase/1.4.2/docs/guide/security/jce/JCERefGuide.html#AppA

String	$util.newEncryptionUtil() Creates a new Encryption Utility object, initially configured with the default values for its parameters: Algorithm = “AES” Text Character Set = “UTF-8” Binary Encoding = “hex”

Given an Encryption Utility object $crypt:

String	$crypt.encryptText(string data) Encrypts the specified plain text, using the currently configured encryption algorithm. The text will be converted to binary according to the currently configured text charset (default is UTF-8). The binary value will then be encrypted, and the result encoded back to a String using the currently configured binary encoding value (default is hex).
String	$crypt.decryptText(string encryptedData) Decrypts an encrypted value and decodes the result of the decryption as plain text. The input String will first be converted to binary using the currently configured binary encoding (hex or base64). The binary value will be decrypted, and the result will be converted back into a text string using the currently configured text charset value (default is UTF-8).
String	$crypt.encryptBinary(string data) Encrypts the binary encoded value, using the currently configured encryption algorithm. The text will be converted to binary according to the currently configured binary encoding (hex or base64). The binary value will then be encrypted, and the result encoded back to a String using the currently configured binary encoding value (default is hex).

String	$crypt.decryptBinary(string encryptedData) Decrypts an encrypted value and returns the result of the decryption as a binary encoded string. The input String will first be converted to binary using the currently configured binary encoding (hex or base64). The binary value will be decrypted, and the result will be converted back into a string using the currently configured binary encoding value (hex or base64).
String	$crypt.generateKey() Randomly generates a 128 bit key (16 byte binary, 32 character hex). The value returned will be the binary key encoded in the currently configured binary encoding format.
String	$crypt.generateKey(int keySizeInBits) Randomly generates key of the specified size in bits. The value returned will be the binary key encoded in the currently configured binary encoding format.

Usage:

- **data** – the plain text to be encrypted
- **encryptedData** – the encrypted data
- **keySizeInBits** – size for key measured in bits

Parameters

```
$crypt.algorithm
```

- Returns the algorithm to use

```
$crypt.setAlgorithm(string algorithm)
```

- Sets the encrypted algorithm to use; allowed values are specified in the JCE documentation

```
$crypt.mode
```

- Returns the mode used in the encryption/decryption operation

```
$crypt.setMode(string mode)
```

- Sets the mode to use in the encryption/decryption operation; allowed values are specified in the JCE documentation

```
$crypt.padding
```

- Returns the padding used in the encryption/decryption operation

```
$crypt.setPadding(string padding)
```

- Sets the padding to use in the encryption/decryption operation; allowed values are specified in the JCE documentation

```
$crypt.key
$crypt.keyAsBase64
```

- Returns the key used for encryption and decryption

```
$crypt.setKey(string key)
$crypt.setKeyAsBase64(string keyAsBase64)
```

- Sets the key used for encryption and decryption

```
$crypt.iv
$crypt.ivAsBase64
```

- Gets the currently configured value of the initialization vector (note this method may be of particular use when the utility is configured in such a way that the encryption may generate its own initialization vector, and that value needs to be retrieved for later use)

```
$crypt.setIv(string ivAsHex)
$crypt.setIvAsBase64
```

- Sets the initialization vector

```
$crypt.textCharset
```

- Returns the character set used when converting plain text to and from binary

```
$crypt.setTextCharset(string charset)
```

- Sets the character set to use when converting plain text to and from binary

```
$crypt.binaryEncoding
```

- Returns the currently configured binary encoding format; possible values are "hex" or "base64" (default is "hex")

```
$crypt.setBinaryEncoding(string encoding)
```

- Sets the binary encoding used to transform the output of encryption operations or the input of decryption operations (note encryption algorithms only operate on binary data, but the WCM system deals primarily with string values - as such, binary data must be encoded in either hexadecimal or base64 encoding; permitted values to set are "hex" or "base64" (the default value for binary encoding is "hex")

Print to PDF Tags

Prior to using these tags, minor configuration steps are necessary; please refer to the User Community for directions.

Once configured, to create a link from a content item detail template to view the content item as a PDF, use $cms.content.pdfUrl. For example:

```
<a href="$cms.content.pdfUrl">Print to PDF</a>
```

For fields that utilize an HTML editor, generic XSL transformation utilities will need to be used to transform the HTML into XSL-FO markup for inclusion in the PDF template. HTML needs to be converted to XHTML first, then converted to XSL-FO using the XSLT.

XHTML Fragment	*$cms.xhtml(string htmlInput)* *Converts raw HTML (e.g. from an HTML editor field) into an XHTML fragment.*
XHTML Document	*$cms.xhtmlDocument(string htmlInput)* *Converts the input HTML into a corresponding XHTML document. The output of this method will be a complete XHTML document including the XML declaration, XHTML declaration, the root <html> tag, and <head> and <body> sections.*
String	*$cms.xslt(string xslIncludeFileName, string xmlInput)* *Returns a string of the input translated according to the specified XSL stylesheet.* *Refer to the User Community Code Library for a sample XSL Stylesheet to convert XHTML to XSL-FO.* *In this example the Include is named "html2xslfo":* *#set($xhtml = "$cms.xhtml($cms.content.html)")* *$cms.xslt("html2xslfo"," $xhtml ")*

Usage:

- **xslIncludeFileName** – The stylesheet saved as an Include file.
- **xmlInput** – XML string fragment. Note this fragment should be wrapped in a tag because the $cms.xhtml tag does not guarantee the output will have a single root element.
- **htmlInput** – Raw HTML to be converted.

Assorted Tags

String	*$cms.crumbs(string separator [,boolean hyperlinked] [,string linkStyle, string separatorStyle])*
String	*$cms.crumbs(int destID, string separator, boolean hyperlinked [,string linkStyle, string separatorStyle])* *Outputs breadcrumbs with the specified separator, in the specified styles.*
Integer	*$cms.customerID* *Returns the customerID of the customer whose page is being published. This is an internal identifier which is used in surveys.*
String	*$ip* *In IP Authentication module, returns the user's IP address to be included in a form to request authenticated access.*

Usage:

- **destID** – The destination ID for the website section to build from.
- **separator** – The character used to separate breadcrumb pages such as " | " or " >> "
- **hyperlinked** – true will hyperlink the breadcrumb, false will not.
- **linkStyle** – add styles for the link
- **separatorStyle** – add styles for the separator

Examples:

```
$cms.crumbs(">> ")
```

Will output:

- Sports >> Basketball >> Minor Leagues

Velocity Macros (Velocimacros)

The following Velocity Macros are supported in the Clickability Platform:

```
#toggleMobileSite()
```

In order to allow a Website Visitor to bypass User Agent Rules and receive the "full" or "desktop" version of a given site, use the #toggleMobileSite() macro which contains two functions:

- `viewNonMobileSite()`: sets the User Agent Rules cookie to simulate a desktop browser, rather than a standard browser, disabling the "mobile" view of the site.
- `eraseMobileCookie()`: removes the User Agent Rules cookie entirely, which will result in the user being re-identified for the appropriate device. This effectively re-enables the non-desktop view for the Website Visitor.

Note: because these are JavaScript functions that are produced on pages that are rendered by the Dyanmic Site Platform, and the User Agent Redirect Rules cookie is set for your "www" domain rather than your "mobile" domain, these tags cannot be used to disable mobile redirects when using 3rd party mobile vendors.

```
#commentCode()
#socialMediaCode() and #socialMediaCodeNoLibrary()
```

See the “Comments” manual above for descriptions of these macros.

Other Velocity Macros are not supported for two primary reasons:

1. Due to our shared hosting environment, macros would be limited in scope to individual templates, meaning they would only work within the templates that call them. This limitation dramatically decreases the value of macros.
2. The version of Velocity that Clickability uses has a known bug that causes macros to be unstable; very rarely they will fail to execute leading to incorrectly published pages.

For these reasons, other Velocity Macros are not supported and should not be used.

Debugging Tools

More debugging tools are online at
http://community.clickability.com **> Developer Forum > Debugging and Developer Tools.**

Troubleshooting Modes

URL parameters and Velocity tags can be used to provide insight into the various Clickability Platform components of a page. The following Velocity tags will render the specified output to the page when in debug mode. To enter debug mode, simply append the desired request parameter(s) to the end of the URL you are viewing and reload the page.

Note: Due to the potentially sensitive nature of the data being published, viewing the template debug information requires a login. This is the same login used to access the Clickability Platform.

String	*$codeDebug.log(string message)*
String	$codeDebug.watchpoint()
String	$codeDebug.watchpoint(string var)
String	$codeDebug.watchpoint(arrayList vars)
Integer	$cms.debugInfo.environment
String	$cms.debugInfo.environmentName
Integer	$cms.debugInfo.destinationID
Integer	$cms.debugInfo.destinationVersion
Integer	$cms.debugInfo.templateID
Integer	$cms.debugInfo.templateVersion
Integer	$cms.debugInfo.contentID
Integer	$cms.debugInfo.contentVersion
String	$cms.debugInfo.debugInfo

Usage:

- **message** – The message to be displayed in the Velocity log
- **var or vars** – The variable values to be displayed in the Velocity log.

Example

```
$codeDebug.log("This content item's id is ${cms.content.id} and the title is
${cms.content.title}.")
```

will output the following to the Velocity log:

[Template Name] - INFO – This content item's id is 456234 and the title is How to Build a Raft.

Example

```
$codeDebug.watchpoint()
$codeDebug.watchpoint("content")
$codeDebug.watchpoint(["adModuleItem","content"])
```

will output the following link in the Velocity log:

[Template Name] - Watchpoint 1

Click on this link to see a popup containing information about the specified (or by default: all) variables that exist on the page displayed in the format below:

```
# 1 of 1
Watchpoint: Watchpoint 1
Template Name: Article Detail
Watched Vars: All variables in scope
```

Variable Name	Variable Value
adModuleItem	{Content: Id=546234, Version=3, Type=Article, Type Id=1234, Type Version=4}
brandCatSetID	6789
content	{Content: Id=888456, Version=18, Type=Article, Type Id=3456, Type Version=18}
curDate	Tue Aug 11 21:02:48 PDT 2009
curFullURL	http://www.mysite.com/news/sports/888456.html?debug=full
curSection	{Website Section: Id=12345, Version=14, Type=Site-Section, Type Id=7777, Type Version=21}

Debug Modes

`debug=y` or `debug=basic` (ex: http://www.mysite.com?debug=basic)

- adds a “Referenced Objects” data table at the bottom of the page listing every Clickability Platform object that goes into building that page, along with its lastModified date.
- inserts START/END comments into the source code around each subtemplate, so you can easily identify what template a given part of a page is coming from.

The following log levels will work with debug modes *extended*, *full* or *extendedstatic*.

`&debugLogLevel=error` – An unexpected error that has occurred during template generation that might require some corrective action.

`&debugLogLevel=fatal` - Similar to error, but on more severe types of template generation errors.

`&debugLogLevel=warn` – Likely to require some investigation but may not affect a template's overall generation.

`&debugLogLevel=info` - This is the default log level. This level if suitable for typical debug usage, and will be used if no debugLogLevel parameter is specified.

`&debugLogLevel=debug` – Use this level on a search results page to see how scoring is determined on all search results. This level includes detailed messages that may prove too much noise for general debugging.

`&debugLogLevel=trace` - Lowest level of template generation error logging - only necessary during deep forensics.

`debug=extended` (ex: http://www.mysite.com?debug=extended)

- adds Velocity Logging that will display errors from the site publishing environment and all $codeDebug messages and watchpoints. Use the optional log level parameters above to adjust this view.

`debug=full` (ex: http://www.mysite.com?debug=full)

- adds in the Template Call Heirarchy which displays a hierarchy of templates called to build a page – including calls made in loops.

`debug=extendedstatic` (ex:http://www.mysite.com?debug=extendedstatic)

- this is the same view as "debug=full" except that the template call tree will be displayed statically without the ability to expand/collapse the template views; this is the preferred mode for sites that use hundreds of templates to build a single page.

`adsdebug=y` (ex: http://www.mysite.com?adsdebug=y)

- pops up an alert box on each ad-containing page indicating which ad placements were preloaded & which were loaded individually.

`clickDebugView=overlay`
(ex: http://www.mysite.com?debug=y&clickDebugView=overlay)

- displays the debug results on top of the website view. There is a clickable option in this view for "transparent" or "opaque".

Help Mode

`showHelp=y` (ex: http://www.mysite.com?showHelp=y)

String	$cms.helpText(string helpText [,int divWidth])

Usage:

- **helpText**– the explanatory text for the help icon
- **popupWidth** – the pixel width of the flyover div in which the help text is to be displayed (250 by default)

Help text embedded via $cms.helpText tags is displayed using our default stylesheet. To override some or all of the default styles, just add an include file named 'helpText.css' overriding the following style class selectors as desired:

```
#cmPubHelpTextPopup {}
.hint {}
.hint a {}
```

- displays question marks on the page to indicate the existence of help comments (if any) that have been embedded in the templates.

Cache-Busting Mode

`{any text}` (ex: http://www.mysite.com?anytext)

- the addition of parameters forces the page to be rebuilt by the publisher based on the latest Clickability Platform objects vs. just pulling the cached version.

Dynamic Error Pages

Note: these tags are used in conjunction with the "Enable Dynamic Error Page" checkbox within Domain > Environment configurations.

Given an Error object $httpError:	
Integer	$httpError.statusCode
String	$httpError.requestedUrl
String	$httpError.message
String	$httpError.servletName

Dynamic Error Pages are cached based on the requested URL. A bad URL is negatively cached with a TTL of 1 minute. Since Dynamic Error Pages are generated each time a browser requests a missing resource, the best practice is to keep these pages very lightweight. This means:

- No more than one search
- No more than 10 total template calls (including iteration)
- No more than 100 Instantiated objects
- No calls to external 3rd parties

It is also recommended the tags be used with hidden notation or to check if values are returned prior to printing the errors to the screen.

Refer to additional best practices at http://community.clickability.com > Developer Forum > Technical Docs & Webinars.

Performance

Best practices for Clickability Platform Website Developers are online at http://community.clickability.com > Developer Forum > Technical Docs & Webinars.

Caching

Clickability Platform production sites utilize a caching server to optimize the load on the publishers to prevent having to rebuild a web page from scratch every time the page is requested. Instead, the *first* time a given page is requested on the live site, the publisher builds it based on all its component Clickability Platform objects. It then caches that page on the caching server so that subsequent requests for the same page can be served directly from that cache.

After 5 minutes have passed since that page was stored on the caching server, subsequent requests will consult the publisher and see if any of the component Clickability Platform objects have changed. If so, it will rebuild the page. If not, it will continue to serve from cache. To provide optimal performance, additional caching takes place on our search servers. Search results for search queries will be cached on the search server for up to 20 minutes.

Pages which have a querystring (a question-mark-preceded string at the end of the url: http://www.site.com?abc=xyz) are an exception. These pages do NOT get placed on the caching server and are built fresh with each request.

Implications of Caching Rules

Latency of changes appearing on production:
Changes to content items may take up to 5 minutes to be reflected on the production site. Because of the additional caching on the search servers, new and modified content changes may not appear in search results for up to 20 minutes. (By contrast, the dev and stage sites will reflect changes immediately because they do not use caching.) Appending a random querystring to production pages will "cache-bust" the page and show the most recent changes (not applicable to search results).

Action needed to avoid certain stale scenarios:
As noted above, when any of the Clickability Platform objects used on a page get updated, that page gets rebuilt.

There are some template tags that work best if they dynamically update each time the page loads, but they are not linked to Clickability Platform objects.

Examples:

```
$util.now, $util.tomorrow, $util.yesterday
```

If these tags are on your page, either make sure the page is accessed with a unique querystring so that it's built fresh every time, or ask Client Services to make sure the specified template gets rebuilt on a regular schedule (e.g. once nightly to make sure the date tags reset each night).
Non-caching of any page that has a querystring:
On pages like poll results pages, it's beneficial that the resulting page not be cached (to ensure the query is run fresh every time). But for other pages, like an Article page that may simply use a ?popup=y querystring to indicate a display style, caching of the page is still desirable to avoid having to build that Article page fresh on every request. In such cases, a c=y parameter should be appended to the querystring to force the page to cache, even though it has a querystring.

Example:
http://www.mysite.com/news/1234.html?popup=y&c=y

Robots

Every Clickability Platform website has a robots file set up to keep web crawlers out of places they shouldn't be crawling. For instance, crawling of search pages is generally unnecessary and performance-inhibiting, so by default we have set up the robots files to disallow media and access to any pages located at /search or /searchresults.
If you need to locate your searches at other paths besides the above, or have any other performance-heavy pages that should be kept away from crawlers, ask Client Services how to customize your robots file.

Connections to other Servers

Certain Velocity tags allow you to connect to another server to pull in content to your pages. While appropriate measures have been built in to close the connection in case of a failure, care should still be taken when using these tags since if that server is unresponsive, it could hang the connection and disrupt processes on the Clickability Platform servers.

```
$rss.getChannel(url)
$util.getURL(url)
$util.postURL(url)
$httpClient.newHttpClient()
$soap.sendmySoapRequest()
```

High Load Operations

While the Clickability Platform has been optimized for efficiency, there are still certain operations which when used in unexpected volume can cause your pages to load more slowly. For instance, using $cms.links(1,100) to retrieve the first 100 content items from a placement list might be a perfectly fast operation. However,$cms.links($cat,1,100) could be noticeably slower, especially if the $cat-categorized items you 're looking for are not the first 100 in the placement list. In the latter case, it *might* be more efficient to use search tags, depending on your exact needs.

More best practices for Clickability Platform Website Developers are online at http://community.clickability.com > Developer Forum > Technical Docs & Webinars.

Online Resources

For code examples, technical documentation, and access to connect with other Velocity developers using the Clickability Platform, please visit our User Community at **http://community.clickability.com** > **Developer Forum**.

Appendix A

In July 2010 the Java syntax in this manual was converted to Velocity syntax. The list below contains the Java syntax that was previously published. Please refer to the corresponding Velocity city for further reference.

- .getActions() refer to .actions
- .getActionDefinitions() refer to .actionDefinitions
- .getAllFeedback() refer to .allFeedback
- .getAllThreadedFeedback() refer to .allThreadedFeedback
- .getArticles() refer to .articles
- .getAttribute() refer to .attribute
- .getAudio() refer to .audio
- .getAverage() refer to .average
- .getBlogs() refer to .blogs
- .getCategories() refer to .categories
- .getCategorization() refer to .categorization
- .getCategorizationString() refer to .categorizationString
- .getCategorySet() refer to .categorySet
- .getChildNodes() refer to .childNodes
- .getCommentLength() refer to .commentLength
- .getCommentPeriod() refer to .commentPeriod
- .getComments() refer to .comments
- .getContentDestID() refer to .contentDestID
- .getContentID() refer to .contentID
- .getCount() refer to .count
- .getDestID() refer to .destID
- .getDisplayLevels() refer to .displayLevels
- .getFields() refer to .fields
- .getFirstActivity() refer to .firstActivity
- .getFirstChild() refer to .firstChild
- .getGatedContent() refer to .gatedContent
- .getHasChildren() refer to .hasChildren
- .getID() refer to .id
- .getInterestDefinitions() refer to .interestDefinitions
- .getItem() refer to .item
- .getLeadScoreAdjustments() refer to .leadScoreAdjustments
- .getLength() refer to .length
- .getLevel() refer to .level
- .getMaxDate() refer to .maxDate
- .getMaxLevel() refer to .maxLevel

- .getMedia() refer to .media
- .getMinDate() refer to .minDate
- .getName() refer to .name
- .getNumResponses() refer to .numResponses
- .getNumTotalResponses() refer to .numTotalResponses
- .getOptions() refer to .options
- .getOutboundFeeds() refer to .outboundFeeds
- .getPageviews() refer to .pageviews
- .getParams() refer to .params
- .getParent() refer to .parent
- .getParentID() refer to .parentID
- .getPartner() refer to .partner
- .getProfileFields() refer to .profileFields
- .getProfileLevelForGatedContent() refer to .profileLevelForGatedContent
- .getProfileLevels() refer to .profileLevels
- .getRatings() refer to .ratings
- .getResults() refer to .results
- .getRoot() refer to .root
- .getRootCommentID() refer to .rootCommentID
- .getSelectResult() refer to .selectResult
- .getSetIDs() refer to .setIDs
- .getSets() refer to .sets
- .getStateDefinitions() refer to .stateDefinitions
- .getStatus() refer to .status
- .getTextContent() refer to .textContent
- .getThreadCreateDate() refer to .threadCreateDate
- .getThreadedComments() refer to .threadedComments
- .getThreadedRatings() refer to .threadedRatings
- .getThreadID() refer to .threadID
- .getThreadModifiedDate() refer to .threadModifiedDate
- .getTitle() refer to .title
- .getTotal() refer to .total
- .getUnsetFieldsForGatedContent() refer to .unsetFieldsForGatedContent
- .getUrl() refer to .url
- .getVideo() refer to .video

Appendix B

The following lists are for use with the $util.getLocale template tag.

List of All Locales

Locale Name	Locale Code
Albanian (Albania)	sq_AL
Albanian	sq
Arabic (Algeria)	ar_DZ
Arabic (Bahrain)	ar_BH
Arabic (Egypt)	ar_EG
Arabic (Iraq)	ar_IQ
Arabic (Jordan)	ar_JO
Arabic (Kuwait)	ar_KW
Arabic (Lebanon)	ar_LB
Arabic (Libya)	ar_LY
Arabic (Morocco)	ar_MA
Arabic (Oman)	ar_OM
Arabic (Qatar)	ar_QA
Arabic (Saudi Arabia)	ar_SA
Arabic (Sudan)	ar_SD
Arabic (Syria)	ar_SY
Arabic (Tunisia)	ar_TN
Arabic (United Arab Emirates)	ar_AE
Arabic (Yemen)	ar_YE
Arabic	ar
Belarusian (Belarus)	be_BY
Belarusian	be
Bulgarian (Bulgaria)	bg_BG
Bulgarian	bg
Catalan (Spain)	ca_ES
Catalan	ca
Chinese (China)	zh_CN
Chinese (Hong Kong)	zh_HK
Chinese (Singapore)	zh_SG
Chinese (Taiwan)	zh_TW
Chinese	zh
Croatian (Croatia)	hr_HR
Croatian	hr
Czech (Czech Republic)	cs_CZ
Czech	cs
Danish (Denmark)	da_DK
Danish	da
Dutch (Belgium)	nl_BE
Dutch (Netherlands)	nl_NL
Dutch	nl
English (Australia)	en_AU
English (Canada)	en_CA
English (India)	en_IN
English (Ireland)	en_IE
English (Malta)	en_MT
English (New Zealand)	en_NZ
English (Philippines)	en_PH
English (Singapore)	en_SG
English (South Africa)	en_ZA
English (United Kingdom)	en_GB
English (United States)	en_US
English	en
Estonian (Estonia)	et_EE
Estonian	et
Finnish (Finland)	fi_FI
Finnish	fi
French (Belgium)	fr_BE
French (Canada)	fr_CA
French (France)	fr_FR
French (Luxembourg)	fr_LU
French (Switzerland)	fr_CH
French	fr
German (Austria)	de_AT
German (Germany)	de_DE
German (Luxembourg)	de_LU
German (Switzerland)	de_CH
German	de
Greek (Cyprus)	el_CY
Greek (Greece)	el_GR
Greek	el
Hebrew (Israel)	iw_IL
Hebrew	iw
Hindi (India)	hi_IN
Hungarian (Hungary)	hu_HU
Hungarian	hu
Icelandic (Iceland)	is_IS

Icelandic	is
Indonesian (Indonesia)	in_ID
Indonesian	in
Irish (Ireland)	ga_IE
Irish	ga
Italian (Italy)	it_IT
Italian (Switzerland)	it_CH
Italian	it
Japanese (Japan)	ja_JP
Japanese (Japan,JP)	ja_JP_JP
Japanese	ja
Korean (South Korea)	ko_KR
Korean	ko
Latvian (Latvia)	lv_LV
Latvian	lv
Lithuanian (Lithuania)	lt_LT
Lithuanian	lt
Macedonian (Macedonia)	mk_MK
Macedonian	mk
Malay (Malaysia)	ms_MY
Malay	ms
Maltese (Malta)	mt_MT
Maltese	mt
Norwegian (Norway)	no_NO
Norwegian (Norway,Nynorsk)	no_NO_NY
Norwegian	no
Polish (Poland)	pl_PL
Polish	pl
Portuguese (Brazil)	pt_BR
Portuguese (Portugal)	pt_PT
Portuguese	pt
Romanian (Romania)	ro_RO
Romanian	ro
Russian (Russia)	ru_RU
Russian	ru
Serbian (Bosnia and Herzegovina)	sr_BA
Serbian (Montenegro)	sr_ME
Serbian (Serbia and Montenegro)	sr_CS

Serbian (Serbia)	sr_RS
Serbian	sr
Slovak (Slovakia)	sk_SK
Slovak	sk
Slovenian (Slovenia)	sl_SI
Slovenian	sl
Spanish (Argentina)	es_AR
Spanish (Bolivia)	es_BO
Spanish (Chile)	es_CL
Spanish (Colombia)	es_CO
Spanish (Costa Rica)	es_CR
Spanish (Dominican Republic)	es_DO
Spanish (Ecuador)	es_EC
Spanish (El Salvador)	es_SV
Spanish (Guatemala)	es_GT
Spanish (Honduras)	es_HN
Spanish (Mexico)	es_MX
Spanish (Nicaragua)	es_NI
Spanish (Panama)	es_PA
Spanish (Paraguay)	es_PY
Spanish (Peru)	es_PE
Spanish (Puerto Rico)	es_PR
Spanish (Spain)	es_ES
Spanish (United States)	es_US
Spanish (Uruguay)	es_UY
Spanish (Venezuela)	es_VE
Spanish	es
Swedish (Sweden)	sv_SE
Swedish	sv
Thai (Thailand)	th_TH
Thai (Thailand,TH)	th_TH_TH
Thai	th
Turkish (Turkey)	tr_TR
Turkish	tr
Ukrainian (Ukraine)	uk_UA
Ukrainian	uk
Vietnamese (Vietnam)	vi_VN
Vietnamese	Vi

Clickability by upland

List of All Time Zones

Time Zone/ID

Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek
America/Adak
America/Anchorage
America/Anguilla
America/Antigua
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/ComodRivadavia
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos
America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia
America/Aruba
America/Asuncion
America/Atikokan
America/Atka
America/Bahia_Banderas
America/Bahia
America/Barbados
America/Belem
America/Belize
America/Blanc-Sablon
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago
America/Chihuahua
America/Coral_Harbour
America/Cordoba
America/Costa_Rica
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson_Creek
America/Dawson
America/Denver
America/Detroit
America/Dominica
America/Edmonton
America/Eirunepe
America/El_Salvador
America/Ensenada
America/Fort_Wayne

America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
America/Hermosillo
America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac
America/Indianapolis
America/Inuvik
America/Iqaluit
America/Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Knox_IN
America/La_Paz
America/Lima
America/Los_Angeles
America/Louisville
America/Maceio
America/Managua
America/Manaus
America/Marigot
America/Martinique
America/Matamoros
America/Mazatlan
America/Mendoza
America/Menominee
America/Merida
America/Metlakatla
America/Mexico_City
America/Miquelon
America/Moncton
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau
America/New_York
America/Nipigon
America/Nome
America/Noronha
America/North_Dakota/Beulah
America/North_Dakota/Center
America/North_Dakota/New_Salem
America/Ojinaga
America/Panama
America/Pangnirtung
America/Paramaribo
America/Phoenix
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre
America/Porto_Velho
America/Puerto_Rico
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina
America/Resolute
America/Rio_Branco
America/Rosario
America/Santa_Isabel
America/Santarem
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/Shiprock
America/Sitka
America/St_Barthelemy
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas
America/St_Vincent
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Tijuana
America/Toronto
America/Tortola
America/Vancouver
America/Virgin
America/Whitehorse
America/Winnipeg
America/Yakutat
America/Yellowknife
Antarctica/Casey
Antarctica/Davis
Antarctica/DumontDUrville
Antarctica/Macquarie
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/Palmer
Antarctica/Rothera
Antarctica/South_Pole
Antarctica/Syowa
Antarctica/Vostok
Arctic/Longyearbyen
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe

Asia/Ashgabat
Asia/Ashkhabad
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Choibalsan
Asia/Chongqing
Asia/Chungking
Asia/Colombo
Asia/Dacca
Asia/Damascus
Asia/Dhaka
Asia/Dili
Asia/Dubai
Asia/Dushanbe
Asia/Gaza
Asia/Harbin
Asia/Ho_Chi_Minh
Asia/Hong_Kong
Asia/Hovd
Asia/Irkutsk
Asia/Istanbul
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Kashgar
Asia/Kathmandu
Asia/Katmandu
Asia/Kolkata
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuching
Asia/Kuwait
Asia/Macao
Asia/Macau
Asia/Magadan
Asia/Makassar
Asia/Manila
Asia/Muscat
Asia/Nicosia
Asia/Novokuznetsk
Asia/Novosibirsk
Asia/Omsk
Asia/Oral
Asia/Phnom_Penh
Asia/Pontianak
Asia/Pyongyang
Asia/Qatar
Asia/Qyzylorda
Asia/Rangoon
Asia/Riyadh87
Asia/Riyadh88
Asia/Riyadh89
Asia/Riyadh
Asia/Saigon
Asia/Sakhalin
Asia/Samarkand
Asia/Seoul
Asia/Shanghai
Asia/Singapore
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Tel_Aviv
Asia/Thimbu
Asia/Thimphu
Asia/Tokyo
Asia/Ujung_Pandang
Asia/Ulaanbaatar
Asia/Ulan_Bator
Asia/Urumqi
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yekaterinburg
Asia/Yerevan
Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Faroe
Atlantic/Jan_Mayen
Atlantic/Madeira
Atlantic/Reykjavik
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley
Australia/ACT
Australia/Adelaide
Australia/Brisbane
Australia/Broken_Hill
Australia/Canberra
Australia/Currie
Australia/Darwin
Australia/Eucla
Australia/Hobart
Australia/LHI
Australia/Lindeman
Australia/Lord_Howe
Australia/Melbourne
Australia/NSW
Australia/North
Australia/Perth
Australia/Queensland
Australia/South
Australia/Sydney
Australia/Tasmania
Australia/Victoria
Australia/West
Australia/Yancowinna
Brazil/Acre
Brazil/DeNoronha

Brazil/East
Brazil/West
Canada/Atlantic
Canada/Central
Canada/East-Saskatchewan
Canada/Eastern
Canada/Mountain
Canada/Newfoundland
Canada/Pacific
Canada/Saskatchewan
Canada/Yukon
Chile/Continental
Chile/EasterIsland
Cuba
Egypt
Eire
Etc/GMT+0
Etc/GMT+10
Etc/GMT+11
Etc/GMT+12
Etc/GMT+1
Etc/GMT+2
Etc/GMT+3
Etc/GMT+4
Etc/GMT+5
Etc/GMT+6
Etc/GMT+7
Etc/GMT+8
Etc/GMT+9
Etc/GMT-0
Etc/GMT-10
Etc/GMT-11
Etc/GMT-12
Etc/GMT-13
Etc/GMT-14
Etc/GMT-1
Etc/GMT-2
Etc/GMT-3
Etc/GMT-4
Etc/GMT-5
Etc/GMT-6
Etc/GMT-7
Etc/GMT-8
Etc/GMT-9
Etc/GMT0
Etc/GMT
Etc/Greenwich
Etc/UCT
Etc/UTC
Etc/Universal
Etc/Zulu
Europe/Amsterdam
Europe/Andorra
Europe/Athens
Europe/Belfast
Europe/Belgrade
Europe/Berlin
Europe/Bratislava
Europe/Brussels
Europe/Bucharest
Europe/Budapest
Europe/Chisinau
Europe/Copenhagen
Europe/Dublin
Europe/Gibraltar
Europe/Guernsey
Europe/Helsinki
Europe/Isle_of_Man
Europe/Istanbul
Europe/Jersey
Europe/Kaliningrad
Europe/Kiev
Europe/Lisbon
Europe/Ljubljana
Europe/London
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Mariehamn
Europe/Minsk
Europe/Monaco
Europe/Moscow
Europe/Nicosia
Europe/Oslo
Europe/Paris
Europe/Podgorica
Europe/Prague
Europe/Riga
Europe/Rome
Europe/Samara
Europe/San_Marino
Europe/Sarajevo
Europe/Simferopol
Europe/Skopje
Europe/Sofia
Europe/Stockholm
Europe/Tallinn
Europe/Tirane
Europe/Tiraspol
Europe/Uzhgorod
Europe/Vaduz
Europe/Vatican
Europe/Vienna
Europe/Vilnius
Europe/Volgograd
Europe/Warsaw
Europe/Zagreb
Europe/Zaporozhye
Europe/Zurich
GMT0
GMT
Greenwich
Hongkong
Iceland
Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives

Indian/Mauritius
Indian/Mayotte
Indian/Reunion
Iran
Israel
Jamaica
Japan
Kwajalein
Libya
Mexico/BajaNorte
Mexico/BajaSur
Mexico/General
Mideast/Riyadh87
Mideast/Riyadh88
Mideast/Riyadh89
Navajo
Pacific/Apia
Pacific/Auckland
Pacific/Chatham
Pacific/Chuuk
Pacific/Easter
Pacific/Efate
Pacific/Enderbury
Pacific/Fakaofo
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu
Pacific/Johnston
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Kwajalein
Pacific/Majuro
Pacific/Marquesas
Pacific/Midway
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Palau
Pacific/Pitcairn
Pacific/Pohnpei
Pacific/Ponape
Pacific/Port_Moresby
Pacific/Rarotonga
Pacific/Saipan
Pacific/Samoa
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
Pacific/Yap
Poland
Portugal
Singapore
Turkey
US/Alaska
US/Aleutian
US/Arizona
US/Central
US/East-Indiana
US/Eastern
US/Hawaii
US/Indiana-Starke
US/Michigan
US/Mountain
US/Pacific-New
US/Pacific
US/Samoa
UTC
Universal
Zulu

Acknowledgements

The Clickability Platform template language is built on the Velocity Template Language developed by the Apache Software Foundation (http://www.apache.org). For a more complete understanding of the Velocity project, please visit http://velocity.apache.org/.