Introduction:

Velocity separates Java code from the web pages, making the web site more maintainable over the long run and providing a viable alternative to Java Server Pages (JSPs) or PHP.

<HTML>

<BODY>

Hello $customer.Name!

<table>

#foreach( $mud in $mudsOnSpecial )

#if ( $customer.hasPurchased($mud) )

<tr>

<td>

$flogger.getPromo( $mud )

</td>

</tr>

#end

#end

</table>

VTL:

The Velocity Template Language (VTL) is meant to provide the easiest, simplest, and cleanest way to incorporate dynamic content in a web page. Even a web page developer with little or no programming experience should soon be capable of using VTL to incorporate dynamic content in a web site.

The following rule of thumb may be useful to better understand how Velocity works: **References begin with $ and are used to get something. Directives begin with # and are used to do something.**

<html>

<body>

#set( $foo = "Velocity" )

Hello $foo World!

</body>

<html>

Comments:

Comments allows descriptive text to be included that is not placed into the output of the template engine.

## This is a single line comment.

#\*

Thus begins a multi-line comment. Online visitors won't

see this text because the Velocity Templating Engine will

ignore it.

\*#

#\*\*

This is a VTL comment block and

may be used to store such information

as the document author and versioning

information:

@author

@version 5

\*#

References:

There are three types of references in the VTL: variables, properties and methods.

1. **Variables**

**#set( $foo = "bar" )**

1. **Properties**

**#set( $data.User = "jon" )**

1. **Methods**

**$customer.getAddress()**

**$data.setUser("jon")**

Array:

all array references are now "magically" treated as if they are fixed-length lists

$myarray.isEmpty()

$myarray.size()

$myarray.get(2)

$myarray.set(1, 'test')

Case Substitution:

$foo

$foo.getBar()

## is the same as

$foo.Bar

$data.setUser("jon")

## is the same as

#set( $data.User = "jon" )

$data.getRequest().getServerName()

## is the same as

$data.Request.ServerName

## is the same as

${data.Request.ServerName}

Directives:

Directives always begin with a #

Like references, the name of the directive may be bracketed by a { and a } symbol.

#set:

The #set directive is used for setting the value of a reference.

#set( $primate = "monkey" )

#set( $customer.Behavior = $primate )

Conditionals:

If / ElseIf / Else:

#if( $foo )

<strong>Velocity!</strong>

#end

#if( $foo < 10 )

<strong>Go North</strong>

#elseif( $foo == 10 )

<strong>Go East</strong>

#elseif( $bar == 6 )

<strong>Go South</strong>

#else

<strong>Go West</strong>

#end

#if ($foo == $bar)

In this case it's clear they aren't equivalent. So...

#else

They are not equivalent and this will be the output.

#end

**## logical AND**

#if( $foo && $bar )

<strong> This AND that</strong>

#end

**## logical OR**

#if( $foo || $bar )

<strong>This OR That</strong>

#end

**##logical NOT**

#if( !$foo )

<strong>NOT that</strong>

#end

Loops:

<ul>

#foreach( $product in $allProducts )

<li>$product</li>

#end

</ul>

<ul>

#foreach( $key in $allProducts.keySet() )

<li>Key: $key -> Value: $allProducts.get($key)</li>

#end

</ul>

<table>

#foreach( $customer in $customerList )

<tr><td>$foreach.count</td><td>$customer.Name</td></tr>

#end

</table>

#foreach( $customer in $customerList )

$customer.Name#if( $foreach.hasNext ),#end

#end

Include:

The #include script element allows the template designer to import a local file, which is then inserted into the location where the #include directive is defined. The contents of the file are not rendered through the template engine.

#include( "one.txt" )

#include( "one.gif","two.txt","three.htm" )

#include( "greetings.txt", $seasonalstock )

Parse:

The #parse script element allows the template designer to import a local file that contains VTL. Velocity will parse the VTL and render the template specified.

#parse( "me.vtl" )

Break:

The #break directive stops any further rendering of the current execution scope. An "execution scope" is essentially any directive with content (i.e. #foreach, #parse, #evaluate, #define, #macro, or #@somebodymacro) or any "root" scope (i.e. template.merge(...), Velocity.evaluate(...) or velocityEngine.evaluate(...)).

Stop:

The #stop directive stops any further rendering and execution of the template. This is true even when the directive is nested within another template accessed through #parse or located in a velocity macro.

Evaluate:

The #evaluate directive can be used to dynamically evaluate VTL. This allows the template to evaluate a string that is created at render time. Such a string might be used to internationalize the template or to include parts of a template from a database.

#set($source1 = "abc")

#set($select = "1")

#set($dynamicsource = "$source$select")

## $dynamicsource is now the string '$source1'

#evaluate($dynamicsource)

Define:

The #define directive lets one assign a block of VTL to a reference.

#define( $block )Hello $who#end

#set( $who = 'World!' )

$block

Velocimacros:

The #macro script element allows template designers to define a repeated segment of a VTL template. Velocimacros are very useful in a wide range of scenarios both simple and complex. This Velocimacro, created for the sole purpose of saving keystrokes and minimizing typographic errors, provides an introduction to the concept of Velocimacros.

#macro( d )

<tr><td></td></tr>

#end

#d()

If we want to put something in that cell, we can alter the macro to allow for a body:

#macro( d )

<tr><td>$!bodyContent</td></tr>

#end

Now, if we call the macro just a bit differently, using #@ before the name and providing a body and #end to the call, then Velocity will render the body when it gets to the $!bodyContent:

#@d()Hello!#end

Math:

Velocity has a handful of built-in mathematical functions that can be used in templates with the set directive.

#set( $foo = $bar + 3 )

#set( $foo = $bar - 4 )

#set( $foo = $bar \* 6 )

#set( $foo = $bar / 2 )

#set( $foo = $bar % 5 )

Range Operator:

First example:

#foreach( $foo in [1..5] )

$foo

#end

Second example:

#foreach( $bar in [2..-2] )

$bar

#end

Third example:

#set( $arr = [0..1] )

#foreach( $i in $arr )

$i

#end

Fourth example:

[1..3]