Velocity

Content

What is Velocity?

Velocity is a Java-based template engine. It permits web page designers to use simple yet powerful script elements that reference methods defined in Java code. Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a site that looks good, and programmers can focus solely on writing top-notch code. 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.

Velocity's capabilities reach well beyond the realm of web sites; for example, it can generate SQL or PostScript from templates. It can be used either as a standalone utility for generating source code and reports, or as an integrated component of other systems. When complete, Velocity will provide template services for the Turbine web application framework.

Velocity-Turbine will provide a template service that will allow web applications to be developed according to a true MVC model.

Where do I get releases?

There is no official release yet, but there will be one shortly. The full source can be retrieved via CVS.

Where do I get nightly builds?

Nightly builds of Velocity can be found here.

How do I contribute, give feedback, fix bugs and so on?

The Velocity project really needs and appreciates any contributions, including documentation help, source code and feedback. Suggested changes should come in the form of source code and/or very detailed and constructive feedback.

- There is a list of Active Developers. Submit some code and get your name added!
- Discussion occurs on the Velocity mailing list.
- Access to the CVS "jakarta-velocity" repository is available both online and with a cvs client.

Coding Standards

Submissions to the Velocity project must follow the coding conventions outlined in this document. Velocity developers are asked to follow coding conventions already present in the code. (For example, if the existing code has the bracket on the same line as the if statement, then all subsequent code should also follow that convention.) Anything not explicitly mentioned in this document should adhere to the official Sun Java Coding Conventions.

Developers who commit code that does not follow the coding conventions outlined in this document will be responsible for fixing their own code.

1. Brackets should begin and end on new lines. Examples:

```
if ( foo )
{
    // code here
}

try
{
    // code here
}
catch (Exception bar)
{
    // code here
}
finally
{
```

```
    // code here
}
while ( true )
{
    // code here
}
```

2. Spaces between parentheses are optional. The preference is to exclude extra spaces. Both of these conventions are acceptable:

```
if (foo)
or
if ( foo )
```

3. Four spaces. NO tabs. Period. The Velocity mailing list receives cvs commit messages that are almost impossible to read if tabs are used.

In Emacs-speak, this translates to the following command: (setq-default tab-width 4 indent-tabs-mode nil)

4. Use Unix linefeeds for all .java source code files. Only platform-specific files (e.g. .bat files for Windows) should contain non-Unix linefeeds.

5. Javadoc MUST exist on all methods. Contributing a missing javadoc for any method, class, variable, etc., will be greatly appreciated as this will help to improve the Velocity project.

6. The Jakarta Apache/Velocity License MUST be placed at the top of every file.

Contributors

The people listed below have made significant contributions to Velocity by working long and hard to make quality software for the rest of the world to use.

If you would like to contribute to Velocity, please see the to do list to find areas where you can contribute. If there is nothing in there that suits your interest, but you still have ideas, please feel free to suggest them on the mailing list.

Velocity follows a certification scheme similar to the one that is outlined on http://www.advogato.org/certs.html. Names on this list are ordered from first contributor at the top to the most recent contributor at the bottom.

Jason van Zyl jvanzyl@periapt.com Periapt Master Jon S. Stevens jon@latchkey.com CollabNet Master Daniel L. Rall dlr@collab.net CollabNet Master Dave Bryson dave@miceda-data.com Miceda-Data Master Josh Lucas josh@stonecottage.com CollabNet Master Bob McWhirter bob@werken.com Werken & Sons Company Master Terence Parr parrt@jguru.com JGuru Master Geir Magnusson Jr. geirm@optonline.net Independent Master

We would also like to make special mention, and give credit for the original idea Velocity is based on, as Velocity is a new implementation of the model/view/controller architecture, concept, and syntax of the WebMacro Servlet Framework (www.webmacro.org) originally envisioned and implemented by Justin Wells at Semiotek Inc.

Design

Velocity is a Java-based template engine. It can be used as a standalone utility for generating source code, HTML, reports, or it can be combined with other systems to provide template services. Velocity will be tightly integrated with the Turbine web application framework. Velocity-Turbine provide a template service by which web applications may be developed according to a true MVC model.

Velocity has a myriad of potential uses -- generating SQL, PostScript, or Java source code from templates -- but it is expected to be most widely used by web developers looking for a viable alternative to PHP and Java Server Pages (JSPs).

Velocity allows web page designers to embed simple yet powerful script elements in their web pages. These script elements work in conjunction with a Context object, which is defined in Java code. A context object--essentially a Hashtable that provides get and set methods for retrieving and setting objects by name within the Context --provides a "hook" from the Java code to the Velocity script elements. These script elements allow a web designer to retrieve objects from the Context and insert these into a web page as text values. The web designer has some control over looping (for each) and conditional statements (if/else).

Velocity enforces a Model-View-Controller (MVC) style of development by separating Java code from HTML template code. Unlike JSPs, Velocity does not allow Java code to be embedded in pages. Unlike PHP, Velocity does not implement features with other functions. The MVC approach is one of Velocity's great strengths, and allows for more maintainable and better designed web pages.

Although MVC-style development can sometimes lead to longer incubation periods for web sites, particularly if the developers involved are new to MVC, this approach saves time over the long term (believe us, we have been doing this for a long time now). The MVC abstraction prevents web page designers from messing with a software engineer's Java code, and programmers from unduly influencing the look of web sites. Velocity enforces a contract that defines what roles people play in the web site development process.

Velocity uses a grammar-based parser generated by JavaCC (Java Compiler Compiler) using the JJTree extension to create an Abstract Syntax Tree. Its concept was borrowed from WebMacro. Those involved in the Velocity project acknowledge and appreciate the development and design work that went into the WebMacro project.

Developer's Guide

This is still in progress. It will be here soon.

Getting Started

This document provides developers with simple documentation for getting started with Velocity. For information about the overall structure of Velocity, please refer to the Design document.

Instructions for downloading and installing Velocity can be found on the Install document.

View Detailed API Documentation

If you are working on Unix/Linux and would like to create a full set of detailed API

documentation for Velocity, go to the "build" directory and run the following script.

```
./build-velocity.sh javadocs
```

If you are working on Windows, the following command will have the same effect.

```
build-velocity.bat javadocs
```

Installation

Everything required to build Velocity comes with the distribution.

Compiling

On Unix/Linux, execute the following script in the velocity/build directory:

```
./build-velocity.sh
```

On Windows, execute the following script in the velocity\build directory:

```
build-velocity.bat
```

Executing this script will create a "bin" directory within the Velocity distribution directory. The bin directory will contain the compiled class files (inside a "classes" directory) as well as a "velocity.jar" file. Be sure to update your classpath to include Velocity's .jar file.

If you get a compiler error related to one or more packages that could not be found, ensure the build-velocity script you are using is edited to properly specify the paths to each of the packages that could not be found.

Testing Your Installation

There are testing scripts in the velocity/examples directory.

Apache Software License

Velocity is released under the Apache Software License listed below:

```
The Apache Software License, Version 1.1

Copyright (c) 1999 The Apache Software Foundation.  All rights
reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
```

User's Guide

This guide is meant to provide an easy way for designers to get aquainted with Velocity. Velocity may be used as a stand-alone servlet framework, or may be used in conjuction with another servlet framework like Turbine, but in both cases a designer will be using the Velocity Template Language (VTL) to incorporate dynamic content into a site design.

The VTL is meant to provide the easiest, simplest, and cleanest way to render dynamic content in a page design.

VTL Reference

Variables

Velocity references its variables in a fashion similar to Perl (i.e. they use a $), but takes advantage of some Java principles that template designers will find easy to use. For example:

```
$foo
$foo.getBar() or $foo.Bar
```

```
$data.getUser("jon") or $data.User("jon")
$data.getRequest().getServerName() or $data.Request.ServerName
```

These examples illustrate alternative uses for the same variables. Velocity takes advantage of Java's introspection and bean features to resolve the variable names to both objects in the Context as well as the objects methods. It is possible to embed variables almost anywhere in your template. These variables can be evaluated.

Everything coming to and from a variable is treated as a String object. If there is an object that represents $foo (such as an Integer object), then Velocity will call its .toString() method to resolve the object into a String.

Conditionals

If / ElseIf / Else Conditionals

The #if statement in Velocity allows for text in the brackets to be included in the text, on the conditional that the if statement is true. For example:

```
#if ($foo)
    <strong>Velocity Rocks!</strong>
#end
```

The variable $foo is evaluated to see if it is a boolean or not null; the content within the brackets becomes the output if the evaluation is true. Unlike in JSP, Velocity does not force web developers to wrap HTML code within an out.println(), or to delve into ugly workarounds to out.println().

An #elseif or #else element can be used with an #if element.

```
#if ($foo)
    <strong>Velocity Rocks!</strong>
#elseif($bar)
    <strong>Velocity Rocks Again!</strong>
#else
    <strong>Velocity Still Rocks!</strong>
#end
```

In this example, if $foo is false, then the output will be Velocity Still Rocks!

Note that logical operators are not yet available in Velocity. This functionality is expected to be added soon. An example of a logical operator is shown below.

```
#if ($foo && $bar)
    <strong>Velocity Rocks!</strong>
#end
```

Loops

Foreach Loop

The #foreach element allows for looping. For example:

```
<ul>
```

```
#foreach ($product in $allProducts)
    <li>$product</li>
#end
</ul>
```

This #foreach loop causes the $allProducts list (the object) to be looped over for all of the products (targets) in the list. Each time through the loop, the value from $allProducts is placed into the $product variable.

The contents of the $allProducts variable is either a Vector, a Hashtable or an Array. The value assigned to the $product variable is a Java Object and can be referenced from a variable as such. For example, if $product was really a Product class in Java, its name could be retrieved by referencing the $product.Name method (ie: Product.getName()).

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 /path/to/file.vm
```

Set

The #set script element allows the template designer to set variables within the Context.

```
#set $name = "Fred"
```

When using the #set directive, the variable on the left side must be prefixed with a $. This provides a consistent syntax for referencing variables in Velocity.

The following script elements have not been implemented.

Comments

There are three comment types that allow the template designer to place descriptive text in templates that is not placed into the output of the template engine.

```
## this is a line comment

#*

This is a multiline
block comment used for
longer descriptions.

*#

#**

This is a VelociDoc comment block and
may be used to store author and versioning
information:

@author Designer Dude
@version 5
```

```
*#
```

Stop

The #stop script element allows the template designer to stop the execution of the template engine and return. This is useful for debugging purposes.

```
#stop
```

Macro

With the #macro script element, the template designer can define a time-saving macro.

```
#macro (row $content) <tr><td>$content</td></tr> #end
```

This establishes a macro called "row", which uses HTML tags to put content into its own table data cell in an HTML table. Having defined the #row macro, the template designer can now call the #row macro by name.

```
<table>
#foreach ($element in $list)
    #row ($element)
#end
</table>
```

Here a newly created #row macro is nested inside a #foreach statement. As the #foreach statement loops through each $element target in the $list object, the #row macro will take the value of $element and put it into its table data cell.