Velocity

Content

What is Velocity?

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

Velocity uses a grammar based parser generated by JavaCC (Java Compiler Compiler) using the JJTree extension to create an Abstract Syntax Tree (AST) which may subsequently be traversed (repeatedly if desired) by a tree walker. The tree walker is implemented using the visitor design pattern. This allows the parsing logic to be separated from the actions performed on the resultant AST. For example there are two visitors that come with Velocity:

the first is a simple implementation that produces a visual tree of the syntax and nothing more, mostly used for debugging; the second is the visitor that actually generates output from user defined values and introspection.

Where do I get releases?

There is no official release yet, but there will be one shortly. But you may retrieve the full source via CVS.

Where do I get nightly builds?

You can find the builds here.

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

We really need and appreciate any contributions you can give. This includes documentation help, source code and feedback. Discussion about changes should come in the form of source code and/or very detailed and well thought out constructive feedback.

- We have a complete list of Active Developers. Submit some code and get your name added!
- We have a Velocity mailing list for discussion.
- Access to the CVS "jakarta-velocity" repository is available both online as well as with a cvs client.

Coding Standards

This document describes a list of coding conventions that are required for code submissions to the project. By default, the coding conventions for most Open Source Projects should follow the existing coding conventions in the code that you are working on. For example, if the bracket is on the same line as the if statement, then you should write all your code to have that convention.

If you commit code that does not follow these conventions and you are caught, you are responsible for also fixing your own code.

Below is a list of coding conventions that are specific to Turbine, everything else not specificially mentioned here should follow the official Sun Java Coding Conventions.

1. Brackets should begin and end on a new line. Examples:

if (foo) { // code here } try { // code here } catch (Exception bar) { // code here } finally { // code here } while (true) { // code here }

2. It is OK to have spaces between the parens or not. The preference is to not include the extra spaces. For example, both of these are ok:

if (foo) or if (foo)

3. 4 spaces. NO tabs. Period. We understand that a lot of you like to use tabs, but the fact of

the matter is that in a distributed development enviroment, when the cvs commit messages get sent to a mailing list, they are almost impossible to read if you use tabs.

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

4. Unix linefeeds for all .java source code files. Other platform specific files should have the platform specific linefeeds.

5. Javadoc MUST exist on all your methods. Also, if you are working on existing code and there currently isn't a javadoc for that method/class/variable or whatever, then you should contribute and add it. This will improve the project as a whole.

6. The Jakarta Apache/Velocity License MUST be placed at the top of each and 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 become a contributor, please see the document titled todo.html 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.

We are following a certification scheme like the one that is outlined here: http://www.advogato.org/certs.html. The ordering of the names is based on the first contributor at the top to the most recent 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

Design

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

The original concept for Velocity was borrowed from WebMacro. We are gratious for the amount of development and design work that went into WebMacro.

The fundamental design of Velocity is around the idea that there are a few useful script elements that you embed within your HTML code that work in conjunction with a Context object that is populated in your Java code. The purpose of the Context object is to provide a "hook" from your Java code to your Velocity code. Essentially a Context object is simply a Hashtable which provides get and set methods for retrieving and setting objects by name within the Context. These script elements provide enough basic functionality to allow you to retrieve objects from the Context and put them into your page as text values with some degree of control over looping (for each) and conditional statements (if/else).

For people who are not familiar with MVC style of development, at first glance, it will appear as though Velocity is missing a large set of functionality. It turns out that this is actually the strength of Velocity. For example, unlike JSP, there is no way to embed Java code within your page and the script elements provide little more than basic looping and conditional statements. Another example is the PHP model where every single feature is implemented with another function. So, how is this better than the alternatives? The answer is that by using Velocity, you are enforcing a MVC style of development which defines that your Java code and your HTML template code should be separate. This style of development sometimes takes slightly longer (especially if you are new to MVC), but results in much more maintainable code over the long term (believe us, we have been doing this for a long time now). It also provides an abstraction that prevents page designers from messing with software engineer's Java code. In other words, it provides an enforcement of a contract that defines what roles people play in the development process.

User's Guide

Place holder for the Developer's Guide.

Getting Started

The purpose of this document is to define simple documentation on getting started with Velocity. For information about the overall structure of Velocity, please refer to the Design Document . The composition of this document will be written from the perspective of what you should do in order to start a newly functional system.

The first thing that you should do is download and install Velocity. This step is documented in the INSTALL document, so it will not be covered here. If you would like to view the detailed API documentation run "build-velocity.sh|bat javadocs" from the "build" directory. This will create a full set of Velocity API docs in the "docs/apidocs" directory.

Installation

Everything required to build Velocity comes with the distribution.

Compiling

Execute the following script in the velocity/build directory:

./build-velocity

Executing the above script will create a "bin" directory within the Velocity distribution directory. In there will be the compiled class files (inside a "classes" directory) as well as a "velocity.jar" file. You can either use the .jar file directly or copy the org directory into your classpath.

If you get compiler error about not being able to find some package, then it means that you have not edited the scripts to properly specify the paths to each of the packages.

Testing Your Installation

There is testing scripts in the velocity/examples directory.

Apache Software License

Velocity is released under the Apache Software License listed below:

Script Elements

There are several different types of Script Elements within Velocity. The overall purpose of these elements is described in the Design Document. The following elements are currently defined within Velocity and also explained in detail below. All of the elements are prefixed with a #. For example, #if, #foreach, #set.

- Variables

- Conditionals:
  - If / Else
- Loops:
  - Foreach
  - While (Not Yet Implemented)
- Parse
- Include
- Param
- Set
- Comment
- Stop

Variables

Variables are referenced in a similar fashion to the way that they are in Perl (ie: they use a $), but they also take advantage of some Java principles that are easy for template designers to work with. For example:

$foo $foo.getBar() or $foo.Bar $data.getUser("jon") or $data.User("jon") $data.getRequest().getServerName() or $data.Request.ServerName

As you can see in a couple of the examples above there are 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 the variables almost anywhere in your template and have them be evaluated.

Everything coming to and from the variables is treated as a String object. If you have an object that is representing $foo such as an Integer object, then Velocity will call its .toString() method in order to resolve the object into a String.

Conditionals

If / Else Conditionals

The #if statement in Velocity allows you to conditionally include the text within the brackets. For example:

#if ($foo) { Velocity Rocks! }

The variable $foo is evaluated to see if it is a boolean or not null and the content within the brackets is what is output if the evaluation is true. As you can see, this has the advantage over other systems because you do not need to wrap your HTML code within an out.println(), therefore enabling you to develop a more MVC solution. Of course there are other solutions to out.println() within JSP, but they are just as ugly as out.println().

Another example is that you can include #else elements with your #if element.

#if ($foo) { Velocity Rocks! } #else { Velocity Still Rocks! }

Note: In the current version of Velocity, it is not possible to do something like what is shown below. We will be adding this functionality quickly though.

#if ($foo && $bar) { Velocity Rocks! }

If you would like to emulate the functionality of #elseif, then you can do so by using nested #if statements. As you would expect, nesting elements works for all of the elements to an infinite level of nesting. For example, you can nest a #foreach within a #if. We will be considering in the future of adding a #elseif to simplify things.

Loops

Foreach Loop

The #foreach element allows you to create loops. For example:

 #foreach $product in $allProducts { $product }

The #foreach loop causes the $allProducts list to be looped over for all of the elements 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, Hashtable or Array. Thus, the value that is assigned to the $product variable is a Java Object and can be referenced from the variable as so. For example, if $product was really a Product class in Java, you could get the name of the Product by referencing the $product.Name method (ie: Product.getName()).

Parse

The #parse script element allows you to import a local file and have it parsed through the Velocity template engine and inserted into the location where the #parse directive is defined. The current Context is applied to the variables that are embedded within the template.

#parse /path/to/file.vm

Include

The #include script element allows you to import a local file that is 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

Param

The #param script element allows the template designer to set items in the context that are static and do not change over the life of the template.

#param $date = "May 24, 1973"

One difference between Velocity and WebMacro is that the left hand side variable must be prefixed with a $. We felt that this is more appropriate for the script element language because you are effectively setting a variable and the references to variables should be consistent.

Set

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

#set $name = "Fred"

One difference between Velocity and WebMacro is that the left hand side variable must be prefixed with a $. We felt that this is more appropriate for the script element language because you are effectively setting a variable and the references to variables should be consistent.

Currently, the above example is the only thing that is supported. We will eventually be suporting the full range of WebMacro functionality here.

Comment

The ## script element allows the template designer to write comments in templates that are not placed into the output of the template engine.

this is a comment

A current problem with Velocity is that the space that comments occupy is not removed from the template. The newlines should be removed. This will be fixed shortly.

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

User's Guide

Place holder for the User's Guide.