Voleso
Programmer's Guide

Scott Augé

First Edition

Amduus Information Works, Inc.
http://www.amduus.com

Voleso: Pogrammer's Guide

Printed in the United States of America.

Published By:

Amduus Information Works, Inc.
1818 Briarwood, Flint, MI 48507
http://www.amduus.com

Printing History:
August 2011: First Edition

Table of Contents

Introduction

The first time one needs to rework the HTML for look and feel in an application, one quickly learns about the need for separating the HTML from the application code. HTML Mapping was an attempt at this, however, there is still the need for a Progress Development License as well re-compiling on changed code.

Other languages have encountered this need – PHP with the Smarty template processor as well the various one's used in the C and C++ world. These processors do not require re-compiling and allow for HTML designers to work outside of the application code environment.

Thus comes Voleso, a tool that allows developers to separate their code from the HTML of the web page. Voleso is influenced by the PHP Smarty template system. This is because of the language some elements cannot be exact (for example display() in Smarty cannot be accomplished because it is a keyword in the ABL), however those familiar with Smarty should be functional with Voleso quickly.

Why Smarty? Because it is expected there will be a lot more designers familiar Smarty than other systems.

Voleso was named by an automatic project naming tool that can be found at http://amduus.com/ProjectNamer/

Installation

Voleso is distributed in a zip file with the format:

voleso.yyyyjjjhhmmss.zip

The date stamp acts as a build number. The values are:

yyyy	Year of build
jjj	Julian day of build
hh	Hour of build
mm	Minute of build
ss	Seconds of build

Within one will find a directory tree:

/doc	Documentation
/src	Directory leading to the com.amduus... file structure.
/schema	Any tables if needed
/script	Any scripts if needed

One should unzip the file and set the PROPATH to include the /src directory. One may want to have multiple versions of the software on for testing purposes. Simply changing the PROPATH can aid in this.

Voleso requires Progress ABL version 10.2B or better.

Loading A Template

The templates are HTML files with special markup that are accessed on the broker machine of the ABL program.

There are multiple ways of loading an HTML template file.

From a file:

The most common way is from a file. One can use the constructor with simply the file name in a PROPATH oriented fashion:

```
define variable S as class
com.amduus.voleso.clsVoleso no-undo.

S = new
com.amduus.voleso.clsVoleso("oas/template/test1.html"
).
S:HTMLRender(stream WebStream:handle).

delete object S.
```

Here one can see the class is instanced with the name of a template file as it's argument.

Another is from the Reset method, which allows the re-use of an object without having to destroy it. (As of this writing, a little expensive in the ABL to destroy and then re-make an instance.)

```
S:Reset("oas/template/test1.html").
```

From HTML text:

To do so from HTML text, be sure the HTML text is of longchar type and simply call the before mentioned methods and constructors. They are overloaded.

Rendering The Template File

Once the template file has been loaded and all the tags processed, one will want to render the file. This is accomplished by sending the handle of the stream you wish to render the file at. The following example renders the file to the WebStream[1] found in the Webspeed product from Progress Software:

```
define variable S as class
com.amduus.voleso.clsVoleso no-undo.

S = new
com.amduus.voleso.clsVoleso("oas/template/test1.html"
).
S:IncludeFile ("oas/template/testinc.html").
S:HTMLRender(stream WebStream:handle).

delete object S.
```

One can render a file also to an operating system stream for email, later web server pick up, etc.

1 WebStream is used by the {&OUT}, etc. predefinitions.

Commenting

Commenting is accomplished by surrounding the text with {* and *}.

These comments are removed from the HTML upon loading it into the class. HTML based comments will remain! Example in bold:

```
<html>
<body>
Test of include
{* This comment should be removed *}
{include name=oas/template/testinc.html }<!-- Should
still be here -->
{section name=1}
Counter is {$Value1}<br>
{/section}
</body>
</html>
```

One deletes comments with the `:RemoveSmartyComments()` method. It will search the template for such comments and delete them. Often one will want to do this before any processing – it helps eliminate the size of the template.

These comments will appear in rendered HTML (since they do not use HTML markup!)

Any comments in HTML markup will be available in the output HTML.

Inserting Files

There are times when one will want to insert HTML files into HTML files. For example, header.html and footer.html files might be useful instead of propagating that code over and over and over.

When processing a file, the programmer should perform the inserts of the files first – this way one can be sure all the other tags are in play before they are replaced with real data.

Inserting By File Name

Inserting files is done with the include tag, which takes as a field the "name" attribute. This name attribute is the PROPATH based name of the file to load into the template.

```
<html>
<body>
Test of include
{* This comment should be removed *}
{include name=oas/template/testinc.html}<!-- Should
still be here -->
{section name=1}
Counter is {$Value1}<br>
{/section}
</body>
</html>
```

Note at the time, no quoting is done on the name, unlike in HTML development.

The ABL used to accomplish this is done through the `:IncludeFile()` method:

```
define variable S as class
com.amduus.voleso.clsVoleso no-undo.

S = new
com.amduus.voleso.clsVoleso("oas/template/test1.html"
).
S:IncludeFile ("oas/template/testinc.html").
S:HTMLRender(stream WebStream:handle).

delete object S.
```

Automatically Inserting Files

One can also include all the include files with the `:AutoIncludeFile()` method. This is helpful for making sure all the templates related to the page are already loaded before doing other substitutions.

```
define variable S as class
com.amduus.voleso.clsVoleso no-undo.

S = new
com.amduus.voleso.clsVoleso("oas/template/test1.html"
).
S:AutoIncludeFile ().
S:HTMLRender(stream WebStream:handle).

delete object S.
```

This is a recursive algorithm, so one can include files within included files. Beware of circular includes however!

Simple Variable Replacement

One of the main purposes of the template system is to insert data where there are data portions. This is done with the **{&var}** or the webspeed delimiters `` tag. (Note with the webspeed delimiters these are no longer associated with an PUT statement, so they MUST be simply the name of the variable. The programmer decides the format.)

An example in the template file is bolded below:

```
<p>This is testinc.html {$Value}</p>
```

or

```
<p>This is testinc.html `Value`</p>
```

In this example ABL code, the variable Value will be replaced with Scott.

```
define variable S as class
com.amduus.voleso.clsVoleso no-undo.

S = new
com.amduus.voleso.clsVoleso("oas/template/test1.html"
).
S:IncludeFile ("oas/template/testinc.html").
S:AssignVar("Value", "Scott").
S:HTMLRender(stream WebStream:handle).

delete object S.
```

One can send the following types to `:AssignVar()`:

- LongChar

- String
- Date
- Integer
- Decimal
- Logical

Note that Logical uses the method Logical2String that will return either “yes”, “no”, or “?” as string values! It may not use the values set up in the format of the field. So if you have a field formatted as “Bolted/Unbolted” as the yes/no value, it will NOT return Bolted or Unbolted, but will return yes or no.

The `:AssignVar()` method is overloaded in many ways, allowing for different types as well as formatting for them. Review the source code for the various options available in the clsVoleso.cls file.

In the future, it is planned to include the format specified in the HTML text rather than relying on the programmer.

Sections

There are two purposes to sections: there will come times when the same data type will need to be repeated over and over. Other times they will need to be deleted or shown.

Looping Sections

For example, some data being placed into a table or AJAX based tool. This is a good place to use sections.

Sections are delimited by the section tag. There is a name to the section so as developers will know how to pull the section out when there is more than one.

```
<html>
<body>
Test of include
{* This comment should be removed *}
{include name=oas/template/testinc.html }<!-- Should
still be here -->
{section name=1}
Counter is {$Value1}<br>
{/section}
</body>
</html>
```

In this example, we instance two objects called S and T which are both Voleso classes. S will contain the template file. Later on, when processing a section, T will be used.

We use the `:GetSection()` method in S to pull out a section by the name "1" into T. This way T will have the template code bounded by the section tags.

Since T is a Voleso class object, we can use the same routines for changing out variables, etc. within the section.

Since it is a loop, we need to call `:NextSection()` in T to repeat the template code into the out-going code.

When we are done, we `:ApplySection()` with the results of T where the "1" section can be found. The entire section will be replaced with the activities that occurred on T within S.

Once we have done so, we can either delete the object T or re-use it via the `:Reset()` method.

Finally we use `:HTMLRender()` method to send the template with all it's data filled out to the user.

```
define variable S as class
com.amduus.voleso.clsVoleso no-undo.
define variable T as class
com.amduus.voleso.clsVoleso no-undo.

define variable i as integer no-undo.

S = new
com.amduus.voleso.clsVoleso("oas/template/test1.html"
).
S:IncludeFile ("oas/template/testinc.html").
S:AssignVar("Value", "Scott").

/* Deal with a section */

T = new com.amduus.voleso.clsVoleso (S:GetSection
("1")).

{&OUT} "Template:".
T:HTMLRender(stream webstream:handle).

do i = 1 to 3:
```

```
  T:AssignVar ("Value1", string(i)).
  if i < 3 then T:NextSection().

end. /* do */

S:ApplySection("1", T).

delete object T.


S:HTMLRender(stream WebStream:handle).

delete object S.
```

Showing A Section

There will be times when it is optional to show a section of the template or not. To specifically show a section (without the section delimiters, call the `:ShowSection(NameOfSection)` method. This will leave any computations that have occurred in the section alone, it mostly removes the delimiters for the section.

When using `:ApplySection(NameOfSection, clsVoleso)` one does not need to call `:ShowSection()`.

Deleting A Section

There are times when a section needs to be deleted. Simply call the `:DeleteSection(NameOfSection)` call and the section (and subsections) will be replaced with “”.

It is best to delete sections as soon as possible so the string processing works with the smallest strings possible.

Yanking Sections From A Template

There is a special object called `clsRawSectionYanker`. It will receive either a longchar containing the template it's self or a character containing the path to the file containing the template.

One can use the `:YankSection(NameOfSection)` method to pull back a section complete with it's delimiters. Often this is used for text to be replaced by other text after computing.

If one wants the section without it's delimiters, use the `:YankSectionNoDelimiters(NameOfSection)` method. Often this is the text that is sent to computation to be manipulated.

This object will not offer any methods to manipulate template text, this is the purpose of the `clsVoleso` file.

The `clsVoleso` object makes use of this class for it's own purposes, but it is documented here for completeness.

Troubleshooting

Problem: The screen comes up blank! Or the HTML is all over the place.

Symptoms: Agent replies in the log file of the agent:

```
(Procedure: 'ExtractSection
com.amduus.voleso.clsVoleso' Line:276) ** Starting
position for SUBSTRING, OVERLAY,
etc. must be 1 or greater. (82)
```

Resolution: Often a section tag has the wrong delimiting characters. For example, the one following has a > where a } should be:

```
{section name=utypeism}
      <colgroup align=center span=1 width="50px"  style="font-size:12px"></colgroup>
      <colgroup align=left   span=1 width="200px" style="font-size:12px"></colgroup>
      <colgroup align=left   span=1 width="70px"  style="font-size:12px"></colgroup>
      <colgroup align=left   span=1 width="30px"  style="font-size:12px"></colgroup>
      <colgroup align=left   span=1 width="70px"  style="font-size:12px"></colgroup>
      <tr>
         <th>Account<br><a href="olssel.htm?sort=1&tlast=1&tdir=0" onmouseover="window.status='Cli
         <th><br><a href="olssel.htm?sort=2&tlast=1&tdir=0" onmouseover="window.status='Click to
         <th><br><a href="olssel.htm?sort=3&tlast=1&tdir=0" onmouseover="window.status='Click to
         <th><br><a href="olssel.htm?sort=4&tlast=1&tdir=0" onmouseover="window.status='Click to
         <th><br><a href="olssel.htm?sort=5&tlast=1&tdir=0" onmouseover="window.status='Click to
      </tr>
      {/section>

   {section name=utypeisa}
      <colgroup align=center span=1 width="50px"  style="font-size:12px"></colgroup>
      <colgroup align=left   span=1 width="200px" style="font-size:12px"></colgroup>
      <colgroup align=left   span=1 width="70px"  style="font-size:12px"></colgroup>
     <tr>
         <th>Account<br><a href="olssel.htm?sort=1&tlast=1&tdir=0" onmouseover="window.status='Cli
         <th><br><a href="olssel.htm?sort=2&tlast=1&tdir=0" onmouseover="window.status='Click to
         <th><br><a href="olssel.htm?sort=3&tlast=1&tdir=0" onmouseover="window.status='Click to
      </tr>
      {/section}

<!-- Generated by Webspeed: http://www.webspeed.com/ -->
```

Future

There are many features of Smarty that are not included in this version.

If you add some, please be sure to pass them along to the author at sauge@amduus.com. You will be included in the contributors section! This is often good for job references.

About

Scott Augé

Scott Augé has been programming in Progress since V6. He also develops in C++, Javascript, HTML, among other languages. He has studied Computational Mathematics and Marketing. For more information, consult his linkedin.com profile at http://www.linkedin.com/in/scottauge.

Amduus Information Works, Inc.

Amduus Information Works, Inc. develops enterprise level software for commercial/non-profit organizations and government agencies. Industries/agencies include manufacturing, service, health care, judicial systems, law enforcement, e-commerce, and real estate. The company develops software per specification as well runs it's own Software as a Service (SaaS) applications.