A Coldbox Plugin for ValidateThis! is Now Available

Posted At : July 23, 2009 12:14 AM | Posted By : Bob Silverberg
Related Categories: Coldbox, ColdFusion, ValidateThis

A developer named Jaime Muniz asked me about whether anyone that I knew had configured ValidateThis!, my validation framework for ColdFusion objects, to work with Coldbox. He also suggested that a Coldbox Plugin for VT might be a good idea. Now I've never actually worked with Coldbox, but I've sure heard a lot of good things about it, so I figured I'd mosey on over the the Coldbox Documentation and see if I could determine what would be required to create a plugin for VT.

Much as I expected, the documentation was extensive and easy to follow, and I was able to grok how to create a plugin in a matter of minutes, so I figured I'd give it a whirl. I'm happy to say that a few hours later I know have a working Coldbox Plugin for VT, as well as a very simple sample application demonstrating its usage. I took the existing Coldbox TransferSample application and made a few changes to it to transform it into a VT sample application. You can find the sample application attached to this post in a zip file. If you're a Coldbox user and have any interest in trying out VT, I'd be eager to hear how it goes and what you think of it.

I'm going to attempt to keep this post uncharacteristically short, so I won't get into the details of using VT here. I have written about that extensively on my blog in the ValidateThis! category. What I will do is briefly describe the API of the Coldbox Plugin, so you'll know what you can do with it, and also walk through the few changes I made to the existing TransferSample application to integrate VT.

The ValidateThisCBPlugin API

The plugin provides the following methods:

- *validate()* - which allows you to perform server-side validations on an object.
- *getValidationScript()* - which allows you to generate client-side JavaScript validations for an object.
- *getInitializationScript()* - which is an optional helper method that does some of the JavaScript set up for you (e.g., includes the required jQuery files).
- *getRequiredFields()* - which returns a list of all of the form fields that are required in the given context. This can be used to dynamically display icons to identify to a user which fields are required.
- *addRule()* - which allows you to define a validation rule dynamically using CFML, rather than having to define all (or any) of your rules in an xml file, which is the default approach.
- *getAllContexts()* - which returns a structure that contains all of the validation rules that have been defined for an object. This isn't generally used for application development, but it's interesting and can help with troubleshooting.

ValidateThisCBPlugin Settings

You can configure the Plugin via settings in your coldbox.xml file. The settings available include: *VT_definitionPath*, *VT_TranslatorPath*, *VT_LocaleLoaderPath*, *VT_BOValidatorPath*, *VT_DefaultJSLib*, *VT_JSRoot*, *VT_defaultFormName*, *VT_localeMap*, *VT_defaultLocale*. All settings have defaults within the framework, so you can use the Plugin without specifying any settings at all. In fact, in the sample application there are no settings defined for VT in the coldbox.xml file. The only setting that you may want to specify yourself is *VT_definitionPath*, which tells VT where to find your xml files that define your validation rules. Regarding this location, when designing the Plugin I took a page from Coldbox's *convention over configuration* book and made the default location for these xml files your *model* directory as defined to Coldbox. So if you put your xml files in your model directory (wherever it may be located), VT will be able to find them without any configuration required on your part.

For more information on what the above mentioned settings actually control, please refer to the VT 0.7 Release Notes.

Integrating VT via the Coldbox Plugin

To keep things simple, I only implemented VT for the Edit User feature of the application, so I only had to change two files:

/handlers/users.cfc

```
<cffunction name="doUpdate" access="public" returntype="void" output="false">
 <cfargument name="Event" type="coldbox.system.beans.requestContext">
 <cfset var rc = event.getCollection()>
 <cfset var oUser = "">

 <cfset oUser = instance.transfer.get("users.users",rc.id)>
 <cfset getPlugin("beanFactory").populateBean(oUser)>
 <!--- I added the following code --->
 <cfset rc.VTResult = getMyPlugin("ValidateThisCBPlugin").validate(theObject=oUser,objectType="users.users") />
 <cfif rc.VTResult.getIsSuccess()>
  <!--- This code was here before (without the <cfif>) --->
  <cfset instance.transfer.save(oUser)>
  <cfset getPlugin("messagebox").setMessage("info", "User Updated")>
  <cfset setNextRoute("users/dspUsers")>
 <cfelse>
  <!--- I added this as well --->
  <cfset rc.oUser = oUser>
  <cfset Event.setView("vwEdit")>
 </cfif>
</cffunction>
```

In /handlers/users.cfc I only had to change the *doUpdate* method. After populating the user object I ask the Plugin to validate() it, which returns a Result object to me which I put into the event context. I then check the result to see if the validations passed. If they did, then I execute the same code that was in this method before I made the changes. If the validations did not pass then I place the user object into the event context and redisplay the view.

That is all I had to change to enable server-side validations via VT. Please note that not being at all familiar with Coldbox I have no idea if this is even close to best practices, but it's simple and it works.

I also had to change **/views/vwEdit.cfm**:

```
<cfoutput>
```

```
<script language="javascript">

 function confirmDelete(){

 }

</script>


<!--- I added the following two lines --->

#getMyPlugin("ValidateThisCBPlugin").getInitializationScript(objectType="users.users")#

#getMyPlugin("ValidateThisCBPlugin").getValidationScript(objectType="users.users",formName="addform")#


<div align="center" class="mainDiv">

 <div style="font-size:25px;font-weight:bold" align="left">

 Edit User

 </div>


 <!--- I also added the following conditional and code to display server-side validation failures --->

 <cfif StructKeyExists(rc,"VTResult")>

  <div align="left" style="margin-top:10px;border:1px solid red;background:##ffffff0;padding:10px">

   The following problems were found with your form submission:

   <ul>

    <cfloop array="#rc.VTResult.getFailures()#" index="failure">

     <li>#failure.Message#</li>

    </cfloop>

   </ul>

  </div>

 <cfelse>

  <div align="left" style="margin-top:10px;border:1px solid black;background:##ffffff0;padding:10px">

   Transfer just retrieved the user information for id: <strong>#rc.id#</strong>. You can now make any updates to the

   user.

  </div>

 </cfif>


 <!--- I didn't change anything else --->


 #renderView('tags/menu',true,10)#


 <div style="margin-top:50px;clear:both" align="left">

  <form name="addform" id="addform" action="#event.buildLink('users.doUpdate')#" method="post">

  <input type="hidden" name="id" value="#rc.oUser.getID()#">

  <table width="100%" cellpadding="5" cellspacing="1" style="border:1px solid ##cccccc;">


   <tr>

    <td width="20" align="right" bgcolor="##eaeaea"><strong>ID:</strong></td>

    <td width="100">#rc.oUser.getID()#</td>

   </tr>

   <tr>

    <td width="20" align="right" bgcolor="##eaeaea"><strong>Create Date:</strong></td>

    <td width="100">#rc.oUser.getcreate_date()#</td>

   </tr>


   <tr>

    <td width="20" align="right" bgcolor="##eaeaea"><strong>First Name:</strong></td>

    <td width="100"><input type="text" name="fname" id="fname" size="50" value="#rc.oUser.getfname()#"></td>

   </tr>


   <tr>

    <td width="20"  align="right" bgcolor="##eaeaea"><strong>Last Name:</strong></td>

    <td width="100"><input type="text" name="lname" id="lname" size="50" value="#rc.oUser.getlname()#"></td>

   </tr>


   <tr>
```

```
    <td width="20"  align="right" bgcolor="##eaeaea"><strong>Email:</strong></td>

    <td width="100"><input type="text" name="email" id="email" size="50" value="#rc.oUser.getemail()#"></td>

   </tr>

  </table>


  <div style="margin-top:20px" align="center" >

   <input type="submit" value="Update" />

  </div>

 </form>

 </div>


</div>

</cfoutput>
```

The two lines that I added near the top call the Plugin and ask it to generate some JavaScript for me. The call to *getInitializationScript()* returns some setup script. It basically loads the required JavaScript libraries and does some initial setup. The call to *getValidationScript()* returns the script that defines each individual JavaScript validation. As you can see I'm just outputting them into the body of the page, which works well for the purposes of this demo.

The next block of code that I added takes the VTResult object which may have been returned from the validate() method that was called in the hander and outputs the validation failure messages to the screen in a friendly format. Note that you can dump the contents of rc.VTResult.getFailures() and see that there is other useful metadata in there, so you're not limited to outputting a bunch of error messages in a single location. For example, you could write code to allow each error message to appear beside its corresponding field.

In addition to the two files that I had to change, I also had to add a number of files. I added a /JS folder which contains the JavaScript files required to allow the client-side validations to function. I also added my Plugin, called ValidateThisCBPlugin.cfc to the /plugins folder. Finally, I added a file called users.users.xml to the /model folder. I kept with my standard naming convention wherein I name my xml rule definition file to correspond to the Transfer package and class name, but you can choose to name your xml files in any way you please. By putting the file into /model I'm taking advantage of Coldbox's and the Plugin's convention, which means I don't have to tell the plugin where to find my files. Here's what users.users.xml looks like:

```
<?xml version="1.0" encoding="UTF-8"?>

<validateThis>

 <objectProperties>

  <property name="fname" desc="First Name">

   <rule type="required" />

  </property>

  <property name="lname" desc="Last Name">

   <rule type="required" />

   <rule type="rangelength" contexts="*">

    <param minlength="5" />

    <param maxlength="10" />

   </rule>

  </property>

  <property name="email" desc="Email Address">

   <rule type="required" />

   <rule type="email" failureMessage="Hey, buddy, you call that an Email Address?" />

  </property>

 </objectProperties>

</validateThis>
```

This file defines the following business rules:

- The First Name, Last Name and Email Address properties are all required.
- The Last Name property must be between 5 and 10 characters long.
- The Email Address property must be a valid email address.

Note that those rules will now be enforced on both the client-side and on the server-side whenever a User is edited, simply by making the few changes that I discussed above. Also note that VT supports many more types of validations rules than those used in this sample, many of which are extremely complex. My goal for the framework is to support any validation requirement that comes along.

In addition to the above changes that I made to the sample application, you will also need to have the ValidateThis! framework installed. That is a simple matter of downloading it from the **Riaforge site** and either placing it under your application root or creating a mapping to it.

I hope to have an online version of this sample application at the **www.validatethis.org demo site** in the very near future. In the meantime you can download the full sample from this post, and it is also included in the VT framework download.

If you are interested in learning more about ValidateThis!, there are a number of resources available:

- I recently recorded a presentation for the **Online ColdFusion Meetup**, which is **available here**.
- The code for the framework, including a number of sample applications is available from the Riaforge Site, **validatethis.riaforge.org**.
- A number of online demos can be viewed at **www.validatethis.org**.
- If you have any questions at all about the framework or how to implement and use it, they can be directed to the ValidateThis! Google Group at **groups.google.com/group/validatethis**

Finally, I'd just like to add that as this is my first time working with Coldbox and, obviously, my first attempt at a Coldbox Plugin, there will undoubtedly be room for improvement. I'd be keen to hear any suggestions or feedback that anyone may have.