ValidateThis 0.96 - Not Just For Objects, JSON Metadata and JavaScript Niceties

Posted At : July 5, 2010 4:34 PM | Posted By : Bob Silverberg
Related Categories: ValidateThis

I've just released version 0.96 of ValidateThis, my validation framework for ColdFusion objects. I guess I'm going to have to come up with a new tagline, because, as of this release, ValidateThis is no longer only for objects. This update includes a bunch of new enhancements, the most significant of which is that you can now use VT to validate a structure. That's right, you no longer need to be working with objects to make use of the framework. More details on that enhancement, and others, can be found following the summary of changes:

- You can now use VT to validate a structure, not just an object.
- Metadata can now be supplied in an external JSON file, as an alternative to the standard XML file.
- You can now have multiple forms on the same html page for the same object, with different contexts.
- A John Whish inspired package of enhancements has been added to the jQuery client-side validations.
- A bug fix reported and patched by a user was implemented.

As always, the latest version can be downloaded from the ValidateThis RIAForge site. Details of the enhancements follow:

ValidateThis Without Objects!

Interestingly, to me anyway, it was extremely simple to enhance the framework to allow a developer to validate a simple structure, rather than requiring them to work with objects. Whereas prior to this release you could only perform server-side validations on business objects (e.g., Transfer, Reactor, CF ORM, etc.), you can now simply pass a structure (e.g., the *form* scope) into the framework's *validate()* method and you'll get back a *Result* object with all of your validation failures. This is significant in that it opens up the framework to anyone writing CFML code, whether they choose to use objects or not. In fact, this enhancement is important enough that I'm going to write a separate blog post about it, with more details and examples. If you can't wait for that, I have added a new demo application to the download, called *StructureDemo*, that shows this new feature in action, and provides all the sample code you need to get up and running.

Metadata in JSON format

Personally I like using XML to define my validation metadata. I find that the metadata is sufficiently complex that it lends itself to a structured format such as XML. You can also get some nice code assistance in CFBuilder and CFEclipse when you use the XML schema document (xsd) that I created for VT. I plan to write a post in the future demonstrating how that works. But there are some people who just don't like XML, and VT isn't about forcing anyone to do anything. It was designed to be as flexible as possible, and designed to support alternate forms of metadata. We have been discussing alternate ways of specifying the metadata that VT needs, and the idea of using JSON came up. It was a reasonably simple task to implement this, so I went ahead and did so. This is also laying the groundwork for supporting the ability to specify validation metadata via annotations in your objects themselves.

So, as of this release, if you would prefer to specify your validation rules as a JSON object you may do so. The structure is very similar to the XML format. Here's a sample XML file followed by the corresponding JSON file. We'll start with the XML file:

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

<validateThis xsi:noNamespaceSchemaLocation="validateThis.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <conditions>

  <condition name="MustLikeSomething"

   serverTest="getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0"

   clientTest="$(&quot;[name='LikeCheese']&quot;).getValue() == 0 &amp;&amp; $(&quot;[name='LikeChocolate']&quot;).getValue() == 0;" />

 </conditions>

 <contexts>

  <context name="Register" formName="registerForm" />

  <context name="Profile" formName="profileForm" />

 </contexts>

 <objectProperties>

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

   <rule type="required" />

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

  </property>

  <property name="Nickname">

   <rule type="custom" failureMessage="That Nickname is already taken.  Please try a different Nickname.">

    <param methodname="CheckDupNickname" />

    <param remoteURL="CheckDupNickname.cfm" />

   </rule>

  </property>

  <property name="UserPass" desc="Password">

   <rule type="required" />

   <rule type="rangelength" >

    <param minlength="5" />

    <param maxlength="10" />

   </rule>

  </property>

  <property name="VerifyPassword">

   <rule type="required" />

   <rule type="equalTo" >

    <param ComparePropertyName="UserPass" />

   </rule>

  </property>
```

```
  <property name="FirstName">
   <rule type="required" contexts="Profile" />
  </property>
  <property name="LastName">
   <rule type="required" contexts="Profile" />
   <rule type="required" contexts="Register">
    <param DependentPropertyName="FirstName" />
   </rule>
  </property>
  <property name="LikeOther" desc="What do you like?">
   <rule type="required" condition="MustLikeSomething"
    failureMessage="If you don't like Cheese and you don't like Chocolate, you must like something!">
   </rule>
  </property>
 </objectProperties>
</validateThis>
```

And here's the corresponding JSON file:

```
{"validateThis" : {
 "conditions" : [
  {"name":"MustLikeSomething",
  "serverTest":"getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0",
  "clientTest":"$(&quot;[name='LikeCheese']&quot;).getValue() == 0 &amp;&amp; $(&quot;[name='LikeChocolate']&quot;).getValue() == 0;"
  }
 ],

 "contexts" : [
  {"name":"Register","formName":"registerForm"},
  {"name":"Profile","formName":"profileForm"}
 ],

 "objectProperties" : [
  {"name":"UserName","desc":"Email Address",
   "rules": [
    {"type":"required"},
    {"type":"email","failureMessage":"Hey, buddy, you call that an Email Address?"}
   ]
  },
  {"name":"Nickname",
   "rules" : [
    {"type":"custom","failureMessage":"That Nickname is already taken. Please try another",
     "params":[
      {"methodName":"CheckDupNickname"},
      {"remoteURL":"CheckDupNickname.cfm"}
     ]
    }
   ]
  },
  {"name":"UserPass","desc":"Password",
   "rules" : [
    {"type":"required"},
    {"type":"rangelength",
     "params" : [
      {"minlength":"5"},
      {"maxlength":"10"}
     ]
    }
   ]
```

```
  },
  {"name":"VerifyPassword","desc":"Verify Password",
   "rules" : [
    {"type":"required"},
    {"type":"equalTo",
     "params" : [
      {"ComparePropertyName":"UserPass"}
     ]
    }
   ]
  },
  {"name":"FirstName","desc":"First Name",
   "rules" : [
    {"type":"required","contexts":"Profile"}
   ]
  },
  {"name":"LastName","desc":"Last Name",
   "rules" : [
    {"type":"required","contexts":"Profile"},
    {"type":"required","contexts":"Register",
     "params" : [
      {"DependentPropertyName":"FirstName"}
     ]
    }
   ]
  },
  {"name":"LikeOther","desc":"What do you like?",
   "rules" : [
    {"type":"required","condition":"MustLikeSomething","failureMessage":"If you don't like cheese and you don't like chocolate, you must like something!"}
   ]
  }
 ]
}
```

As you can see the two formats are very similar. If you want to use a JSON file instead of an XML file you just name your file .json or .json.cfm and the framework will pick it up and read the validation rules from it. When the framework is looking for your rules definition files it will use an XML file if it finds it first, and if one if not found it will use a JSON file if one exists. This means you can use a combination of XML and JSON files if you really wanted to do that. You can thank Adam Drew (who, I hope, will one day have a page of some sort to which I can link) and John Whish for the inspiration for this feature, and Adam in particular for contributing a bunch of code, including the sample JSON file that I used for testing and on which I based the new format.

Support for Multiple Forms on the Same HTML Page

A user, Aaron Miller, reported a problem he was having with multiple forms on an html page, where each form was for the same object, but each form used a different context. The JavaScript validation rules that were being generated in this situation weren't working properly. Being an excellent open source citizen, Aaron sent me a patch for the problem in addition to simply reporting it. I was able to take his patch and extend it to a number of other places in the code that needed to be addressed. I also got some help from none other than Mr. Ben Nadel, who helped me make the now somewhat complex JavaScript a bit more readable (no, not by spacing it out over dozens of lines). John Whish also helped me get past a weak brain moment when I was having trouble figuring out why something wasn't working by suggesting that I try upgrading my version of the jQuery Validation plugin, which did the trick.

This is not a common use case - having multiple forms for the same object with different contexts on the same page - but my ultimate goal for the framework is that it should be able to address any and all validation requirements, so I was pleased to be able to support this.

The John Whish Package

Now, now, all of you John Whish groupies get your heads out of the gutter. I'm talking about a bunch of JavaScript enhancements that John was thoughtful enough to suggest. They include:

- The version of jQuery that is bundled with the framework has been updated to 1.4.2.
- The version of the jQuery Validate plugin that is bundled with the framework has been updated to 1.7.
- The jQuery noConflict option is now being used to minimize the chances of conflicts with other JavaScript frameworks on your page.
- A JSIncludes option has been added to the ValidateThis Config struct to allow you to turn off JavaScript includes globally. Prior to this enhancement if you didn't want the framework to generate JS statements to include jQuery and the other required JS files you had to specify that on each and every call to the *getInitializationScript()* method. You can now set JSIncludes = false in the config struct instead.

BOValidator Bug Fix

Martijn van der Woud has become the ValidateThis exterminator, finding, and fixing, yet another bug. This one related to form names and form contexts, and, as it's now fixed, will be of no interest to anyone other than Martijn and myself, so I'll leave it at that.

Once again, the latest code is available from the ValidateThis RIAForge site, and if you have any questions about the framework, or suggestions for enhancements, please send them to the ValidateThis Google Group. I realize that I've added a lot of features to the framework, none of which have been documented yet, so finishing the documentation is a major item on my to do list, which I plan to complete prior to releasing version 1.0 of the framework. If anyone is interested in volunteering to help with that it would be greatly appreciated.

Finally, I'd like to thank Adam, John, Aaron, Ben and Martijn once again for their contributions to the framework.