ValidateThis! - Let's Talk Metadata

Posted At : October 17, 2008 4:54 AM | Posted By : Bob Silverberg
Related Categories: OO Design, ColdFusion, ValidateThis

In this installment of my series about object oriented validations with ColdFusion I'm going to discuss the metadata that drives all of the client and server-side validations, as well as how a developer can describe that metadata to the framework.

We'll start by looking at a sample xml file, the one that is used to drive the demo in fact. I want to make it clear, however, that the framework is in no way dependent on this xml schema. One can create an xml document based on any schema, as long as it includes the necessary metadata. As well, one can load metadata into the framework programmatically, eliminating the need for xml altogether.

```
<validations>
 <properties>
  <property name="UserName" desc="Email Address">
   <rules>
    <rule type="required" contexts="Register,Update" />
    <rule type="email" contexts="Register,Update"
     failureMessage="Hey, buddy, you call that an Email Address?" />
   </rules>
  </property>
  <property name="Nickname">
   <rules>
    <rule type="custom" contexts="Register,Update">
     <params>
      <param methodname="CheckDupNickname" />
     </params>
    </rule>
   </rules>
  </property>
  <property name="UserPass" desc="Password">
   <rules>
    <rule type="required" contexts="Register,Update" />
    <rule type="rangelength" contexts="Register,Update">
     <params>
      <param minlength="5" />
      <param maxlength="10" />
     </params>
    </rule>
    <rule type="equalTo" contexts="Register,Update">
     <params>
      <param ComparePropertyName="VerifyPassword" />
     </params>
    </rule>
   </rules>
  </property>
  <property name="VerifyPassword" desc="Verify Password">
   <rules>
    <rule type="required" contexts="Register,Update" />
   </rules>
  </property>
  <property name="FirstName" desc="First Name">
   <rules>
    <rule type="required" contexts="Update" />
   </rules>
  </property>
  <property name="LastName" desc="Last Name">
   <rules>
    <rule type="required" contexts="Update" />
    <rule type="required" contexts="Register">
     <params>
      <param DependentPropertyName="FirstName" />
     </params>
    </rule>
```

```
  </rules>
 </property>
 <property name="LikeOther" desc="What do you like?">
  <rules>
   <rule type="required" contexts="Register,Update"
    failureMessage="If you don't like Cheese and you don't like Chocolate, you must like something!">
    <params>
     <param ServerCondition="getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0" />
     <param ClientCondition="$(&quot;[name='LikeCheese']&quot;).getValue() == 0 &amp;&amp; $(&quot;[name='LikeChocolate']&quot;).getValue() == 0;" />
    </params>
   </rule>
  </rules>
 </property>
 <property name="HowMuch" desc="How much money would you like?">
  <rules>
   <rule type="numeric" contexts="Register,Update" />
  </rules>
 </property>
 <property name="AllowCommunication" desc="Allow Communication">
 </property>
 <property name="CommunicationMethod" desc="Communication Method">
  <rules>
   <rule type="required" contexts="Register,Update"
    failureMessage="If you are allowing communication, you must choose a communication method.">
    <params>
     <param DependentPropertyName="AllowCommunication" />
     <param DependentPropertyValue="1" />
    </params>
   </rule>
  </rules>
 </property>
</properties>
</validations>
```

Let's walk through the xml document, identifying the metadata as we go. First off you'll notice that we have *properties*, which correspond to the properties of your Business Object (things like UserName, FirstName, LastName, etc.). Each *property* has a *name* attribute, which again corresponds to the name of the property in the Business Object, and a *desc* attribute, which provides a "friendly" description. The *desc* is used to generate validation failure messages.

Within each *property* we define *rules*. These are the actual validation rules that the framework requires. Each *rule* must have a *type* attribute, which describes the validation type (e.g., required, email, custom, etc.). The following attributes are optional for a *rule*:

- *contexts* - If a validation rule only applies to the Business Object within a particular context (e.g., if a User is being registered) a list of contexts may be specified.
- *failureMessage* - A failure message may be specified which will override the default failure message generated by the framework.

Within each *rule* there is an optional *params* element, which contain *param* elements which describe parameters for the *rule* in question. Some rule types require parameters while others do not. The xml document above shows some examples:

- The *custom* rule type requires a *methodname* parameter which contains the name of the method in the Business Object that must be invoked in order to perform the validation.
- The *rangelength* rule type requires both a *minlength* and a *maxlength* parameter which are used to validate the length of the property's value.
- The *equalTo* rule type requires a *ComparePropertyName* parameter which is used to determine whether the property in question is equal to another property.

The *required* rule type has a number of optional parameters, which can be used to make the required rule conditional:

- Specifying only a *DependentPropertyName* parameter indicates that the property in question is only required if another property has been populated. In the example above, the LastName property is only required if the FirstName property has been populated.
- Specifying a *DependentPropertyName* parameter in conjunction with a *DependentPropertyValue* parameter indicates that the property in question is only required if another property has a specific value. In the example above, the CommunicationMethod property is only required if the AllowCommunication property has a value of 1.
- Specifying a *ServerCondition* parameter indicates that the property in question is only required if the condition specified evaluates to *true*. In the example above, the LikeOther property is only required if both the LikeCheese and the LikeChocolate properties have a value of 0. This only applies to server-side validations.
- Specifying a *ClientCondition* parameter also indicates that the property in question is only required if the condition specified evaluates to *true*. In the example above, we are testing the same condition, that is "the LikeOther property is only required if both the LikeCheese and the LikeChocolate properties have a value of 0". The difference here is that we define the actual JavaScript statement that will be executed to determine if this is true. Obviously this only applies to client-side validations.

And that is all of the metadata that is currently used by the framework. Note that the description of possible *param*s above is not exhaustive. Any new validation type can add any number of required and/or optional parameters.

If one chooses to define these rules in an xml file, which is certainly the way I do it, one can format them in any way, as long as all of the required metadata is included. Of course if you create a new schema you'll have to create a new XMLFileReader object, which will be used to replace the default XMLFileReader object included with the framework, but that's dead easy.

As I mentioned above, for those of you who harbour a dislike for xml (**Matt**), there are two methods available to you which will allow you to define all of the validation

rules programmatically:

- The *addRule* method allows you to add a rule, passing in parameters for propertyName, valType, propertyDesc (optional), contexts (optional), failureMessage (optional), and params (optional).
- The *addPropertyRules* method allows you to add multiple rules for a single property, passing in parameters for propertyName, propertyDesc (optional), and an array of structs that represents the data for the rules.

And that's where things stand as of today. What do you think? Have I missed any important metadata that would be required for *you* to define validation rules?