What's New in ValidateThis 0.98 - Part I

Posted At : November 25, 2010 1:56 PM | Posted By : Bob Silverberg
Related Categories: ColdFusion, ValidateThis

As a follow-up to my post announcing the availability of version 0.98 of ValidateThis, which just included a brief summary of the new features, this post will describe some of those features in more detail.

defaultFailureMessagePrefix Configuration Option

When a validation fails the framework adds a failure message to the Result object that is returned to you. Each validation type has a default failure message, and you also have the option of overriding the default failure message using the *failureMessage* attribute of the *rule* element. The default messaged generated by VT are customized based on the type of failure and any parameters that were in place for the rule. For example, if a rule of type *email* which is defined on the *emailAddress* property of your object fails, the message will read "The Email Address must be a valid email address." If you've defined a rule of type *rangeLength* with a *minLength* of 5 and a *maxLength* of 20 on the *password* property of your object, the default failure message will read "The Password must be between 5 and 20 characters.

Some folks weren't keen on the word "The" being prepended to all of these default messages, so we've introduced a configuration option to change that behaviour. The new option is called *defaultFailureMessagePrefix*, and its default value is "The ", which duplicates the existing behaviour of the framework. If you want to, for example, remove the leading "The ", you can simply pass an empty string into the *defaultFailureMessagePrefix* configuration option. For more information on configuring the framework, please refer to the Configuring the Framework page in the online documentation.

Support for Dynamic Parameters

Many rule types accept parameters. For example, the *rangeLength* type accepts parameters for *minLength* and *maxLength*, which allows you to specify how the rule should behave. For the most part it makes sense for the value of those parameters to be a constant. For example, the password property must be between 5 and 15 characters. You would define such a rule like so:

```
<property name="password">
 <rule type="rangeLength">
  <param name="minLength" value="5" />
  <param name="maxLength" value="10" />
 </rule>
</property>
```

There may be times, however, when you want the value of a parameter to be dynamic. For example, let's say you use the new *dateRange* rule type and you want to specify that the date must be between today and 5 days from today. Putting a constant into your rule definition won't work in that case. What you really need is an expression that can be evaluated when the rule is checked. To support use cases such as this (and even more) we've added the notion of *types* to the *parameter* element. The default type is *value*, which is exactly how the *parameter* element behaved prior to this release. The *value* type tells VT to use the value that you specify in your metadata "as is". To support dynamic parameters which can be evaluated at run time, we've added a new type called *expression*, when VT sees the *expression* type it will not just use your parameter value as is, rather it will evaluate whatever has been specified in the parameter's *value* attribute. So to describe the date-based rule as described above, we could specify:

```
<property name="startDate">
 <rule type="dateRange">
  <param name="minDate" value="now()" type="expression" />
  <param name="maxDate" value="dateAdd('d',5,now())" type="expression" />
 </rule>
</property>
```

This would ensure that the *startDate* property contained a date between today and five days from now.

Not only are expressions supported, but there is a third type available which is *property*. The *property* type tells VT that the value specified contains the name of another property. One use case for the *propertyType* would be if you have a *startDate* and an *endDate* property and you want to make sure the *endDate* is *after* the *startDate*. You could do that using the new *futureDate* validation type, like so:

```
<property name="endDate">
 <rule type="futureDate">
  <param name="after" value="startDate" type="property" />
 </rule>
</property>
```

More information on the metadata that can be used to define your validation rules can be found on the Rules Definition File page in the online documentation.

Custom Validation Type Can Use Methods that Return a Boolean

Prior to this release, in order to use the *custom* validation type, you had to create a special method in your object which VT could call, which returned a structure with keys expected by VT. Specifically, your method had to return a struct with an *IsSuccess* key that contained a boolean value, and a *failureMessage* key which returned the failure message to place in the Result object. As of version 0.98, you can now use a method in a *custom* validation type that simply returns a boolean value. This allows you to reuse methods that you may already have in your domain object, and also means that your domain object does not need to be coupled to VT in any way in order to use the *custom* type.

Note that if you do make use of a method that just returns a simple boolean value, you won't be able to assign a custom failure message from within your method, so you'll need to override the default failure message via the *failureMessage* attribute of the *rule* element.

More information on using the custom validation type can be found on the Using the Custom Validation Type page in the online documentation.

getRequiredPropertiesAndDescs() Method Now Available

A new method was added, which is available through the VT Facade object, called *getRequiredPropertiesAndDescs*. This method accepts an optional *context* argument, and will return to you a structure where each key is the name of a required property and the value of each key is that property's description. This can be useful when generating a UI in which you want to identify the required properties to a user.

But Wait, There's More

Those are only some of the changes and new features added to ValidateThis in version 0.98. Stay tuned for more blog posts about other enhancements.

As always, 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.