ValidateThis 0.95 - Enhancements to the Result Object and Client-Side Validations

Posted At : June 25, 2010 12:01 PM | Posted By : Bob Silverberg
Related Categories: ValidateThis

I've just released version 0.95 of ValidateThis, my validation framework for ColdFusion objects. This update includes some community contributions, as well as a number of features that were prompted by community requests. Here's a summary of the changes, followed by the details for each one.

- Numerous enhancements were made to the Result object, as well as the ability to easily substitute your own Result object for the one that is built into the framework, and the ability to automatically inject the Result object into your business object.
- Client-side validations have been enhanced so that missing form fields will not generate JavaScript errors.
- Client-side validation code has been refactored, and includes a fix from Martijn van der Woud for the *equalTo* validation type.
- More refactoring to set the stage for future enhancements.

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

New Result Object Methods for Returning Validation Failures

Four new methods have been added to the Result object to make getting the validation failures that you want, in a format that is friendly, even easier: *getFailuresByField()*, *getFailureMessagesByField()*, *getFailuresByProperty()* and *getFailureMessagesByProperty()*.

1. The *byField* methods return a struct consisting of clientFieldNames, while the *byProperty* methods return a struct consisting of propertyNames. For the most part you'll want to use the *byField* methods as you are generally dealing with a form in your view when calling these methods.
2. The *getFailures* methods return an array of complete failure structs for each property, while the *getFailureMessages* methods return just the failure message strings. If all you need to do is output text, you could use a *getFailureMessages* method, but if you need to know which set of messages corresponds to which field on your form, you'd need to use a *getFailures* method.
3. By default, the *getFailureMessages* methods return arrays of strings, but you can get a single string, containing all of the failure messages for a given property, by passing in an optional *delimiter* argument. The delimiter provided will be used to concatenate the messages together. So, if you wanted to get back a string for each field, that contains all of the failure messages concatenated using a
, you could use: result.getFailureMessagesByField(delimiter="
").
4. All four new methods accept an optional *limit* argument, which, if specified, will limit the number of messages returned per property. So if you only want one failure message returned per field, you could use: result.getFailuresByField(limit=1). This was an enhancement requested by some community members.

Note that the *getFailuresAsStruct()* method has been deprecated in favour of the new methods described above, as they are more full-featured.

Use Your Own Custom Result Object

If you want to have custom methods in the Result object, for example to do custom formatting of the failure messages, you can now create your own Result object and have the framework return that to you, instead of its own built in Result object. Ideally you'd extend the existing Result object, which would give you access to the full API, and just add and/or override methods. You can tell the framework to use your own object using the new *resultPath* key on the ValidateThis Config struct. Simply provide a dot-notation path to your Result object and the framework will use it.

Have the Result Object Injected into Your Business Objects

Some developers, including the esteemed Matt Quackenbush, prefer to have their validation failures accessible from their business object (BO), as opposed to having to use a separate Result object. After validating a BO, they can then ask the BO if it is valid, and can ask the BO to return any validation failures that were reported. This is easily addressed by simply injecting the Result object that is returned by VT into the BO after validation, but that requires an extra step. It was a simple matter to enhance the server-side validations to do this automatically, so that feature is now available.

If you specify the *injectResultIntoBO* key of the ValidateThis Config struct to be *true* then the framework will inject the Result object into your BO at the end of server-side validations. You will be able to access the Result object by calling *getVTResult()* on your BO. You could then add methods to your BO, perhaps via a base object, that can interact with this composed Result object, allowing you to do things like call *getFailures()* on your BO. Note that this is entirely optional, as many people will not want VT monkeying around with their BOs. If you do not specify a value for the *injectResultIntoBO* key (or specify that it is *false*), VT won't do this injection.

Client-Side Validations for Missing Form Fields Will not Cause JavaScript Errors

At the suggestion of Mark Mandel, I have changed the way the client-side validations work so that you can have rules defined for non-existent form fields without causing issues. Pervious to this release, if the framework generated a client-side validation for a property, but your form did not include a field for that property, a JavaScript error was thrown, causing the validations to not work properly. This caused developers to create contexts for the sole purpose of allowing different forms for the same object type. I realized, after chatting with Mark, that this use of contexts was unnecessary, and this simple change could eliminate the need to use contexts for that purpose.

With this change in place, you should be able to define as many rules as you need for a given object, and if a particular form does not include a field for which a rule is defined, that rule will simply be ignored - on the client side. I think this will make defining rules for objects that appear on multiple forms much simpler, and will do away with the need to use contexts in a number of situations.

Client-Side Refactoring

I tightened up some of the client-side code generation which will make it easier for a developer to add new client-side implementations as well as new client-side rules. During this process I included a fix from Martijn van der Woud for the *equalTo* validation type.

Refactoring to Pave the Way for Bigger Enhancements

I have done a lot of refactoring of the framework to set the stage for some enhancements yet to come. I introduced external file readers which will initially allow for support for json format rules definition files, in addition to the standard xml, and will also allow for new formats to be added easily in the future. I also incorporated Peter Bell's LightWire dependency injection tool to replace the manual dependency injection that was becoming a pain to deal with. Note that this does not impact the users of VT in any way. You don't need LightWire, nor do you need to know anything about it. It's strictly for development of the framework itself.

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'd also like to thank Martijn once again for his contribution to the framework, and also Mark and Matt for their valuable suggestions. Adam Drew, Jamie Krug and John Whish have also been instrumental in shaping this release, as well as providing ideas and guidance for what's coming next. It's rewarding to see the excellent community that is beginning to form around ValidateThis. Thanks everyone!